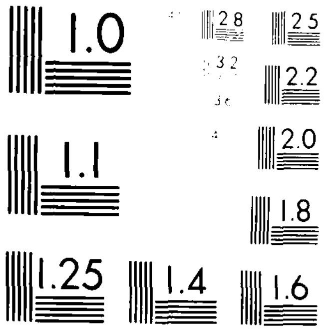
AD-A155 779 AN INTRODUCTION TO ADA (TRADEMARK) FOR SCIENTISTS AND ENGINEERS(U) ARMY MATERIEL SYSTEMS ANALYSIS ACTIVITY ABERDEEN PROVING GROUND MD H E COHEN OCT 83 1/5

UNCLASSIFIED F/G 9/2 NL


1.0
2.8
2.5
3.2
2.2
2.0
1.1
1.8
1.25
1.4
1.6

(11)

AMSAA

AD-A155 779

ARMY MATERIEL SYSTEMS ANALYSIS ACTIVITY

SHORT COURSE:

AN INTRODUCTION TO ADA® FOR SCIENTISTS AND ENGINEERS

12-14 OCTOBER 1983

DTIC FILE COPY

ORGANIZED BY: HERBERT E. COHEN

U S ARMY MATERIEL SYSTEMS ANALYSIS ACTIVITY
ABERDEEN PROVING GROUND, MARYLAND 21005

85

REPORT DOCUMENTATION PAGE

A155 779

Short Course: An Introduction to Ada for Scientists and Engineers

Herbert E. Cohen (Organizer)

Director
U. Army Materiel Systems Analysis Activity
ATTN: DRXSY-MP, APG, MD 21005-5071

Same as Item 9 above.

OCTOBER 1983

UNCLASSIFIED

Approved for public release; distribution is unlimited.

19. SUPPLEMENTARY NOTES

Ada programming, software design, records, arrays, enumeration types, flow of control, program components, design guidelines, record abstraction, numeric abstraction, generics, access types, task and task types, scope and visibility.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Provides an introduction to Ada® programming for engineers, scientists and programmers in the new standard higher order language of the Department of Defense.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SHORT COURSE (TEXT):

AN INTRODUCTION TO ADA®
FOR SCIENTISTS AND ENGINEERS

SPONSORED BY: ADA JOINT PROGRAM OFFICE
3D139 (400 A/N)
THE PENTAGON
WASHINGTON, DC 20301

AND

US ARMY MATERIEL SYSTEMS ANALYSIS ACTIVITY
ABERDEEN PROVING GROUND, MD 21005-5071

ORGANIZED BY: HERBERT E. COHEN
DARCOM MATHEMATICS PROGRAM OFFICE
US ARMY MATERIEL SYSTEMS ANALYSIS ACTIVITY
ABERDEEN PROVING GROUND, MD 21005-5071

ACKNOWLEDGEMENT

The US Army Materiel Systems Analysis Activity (AMSAA) wishes to acknowledge the support provided by the Data General Corporation in making available the MV-4000 computer and the ROLM-Ada compiler for this short course on Ada.

US Army Materiel Systems Analysis Activity would like to express its deep appreciation to LTC Vance Mall (AF) of the Ada Joint Program Office for his continued support throughout the development of this course.

LECTURER FOR

AN INTRODUCTION TO ADA[R]
FOR SCIENTISTS AND ENGINEERS

BILL CARLSON

CENTEC CORPORATION
MANAGEMENT SYSTEMS DIVISION
11260 ROGER BACON DRIVE
RESTON, VIRGINIA 22090-5281

PER CALL JC

A-1

LECTURER'S PREAMBLE

The Ada language brings together 30 years of computer science in a surprisingly well integrated and coherent package. There exist ways to use Ada which are relatively easy to learn and which give you all the power to conventional languages like FORTRAN. This course starts from that direction, so that you are writing Ada programs as quickly as possible. Then we start the process of exploring the broader capabilities of Ada. You will most fully appreciate Ada when you apply it to large programs.

Bill Carlson
October 1983

AN INTRODUCTION TO Ada FOR SCIENTISTS AND ENGINEERS

This course has been designed as a practical introduction to Ada programming and software design for practicing engineers, mathematicians, operations research analysts, statisticians and other professionals. The objective is to communicate the essence of Ada so that students leave confident that they can use Ada effectively. Specific concepts to be taught are the following:

- There is a way to do everything in Ada that you can do in FORTRAN.
- Ada satisfies the requirements which originally caused the DoD to develop a common language.
- Ada programs are structured as one or more packages.
- The Ada compiler can help you write correct programs if you tell it the "type" of each piece of data, and Ada provides a variety of tools for defining new data types, operations on those data types, and controlling the internal representation of data types.
- Generics are used when the same or similar operations are required for several different data types.
- Tasks and exception handling allow asynchronous events to be created and/or modeled.
- Separate compilation is essential for the construction of large systems, and is provided by Ada.

An important goal is to become comfortable with the mechanical aspects of writing an Ada program. This course explains how to write Ada statements, use Ada's control structures, do input/output, and use the Ada development system.

The development system is a Data General Eclipse with the Rolm Ada compiler. That compiler has been validated by the Ada Joint Program Office. The course will introduce the general concept of an Ada Program Support Environment (APSE) and distinguish between concepts which are unique to the particular Data General implementation and those which should be true of all Ada implementations.

ACQUIRING VIDEO TAPES

Title of Tape: "Ada® Programming Language"

1. DoD organizations can obtain free copies of tapes and text by writing to:

Commander
Tobyhanna Army Depot
DAVA
ATTN: DAVA-TLW
Warehouse #3, Bay #3
Tobyhanna, PA 18466

Tapes will be in standard DoD 3/4 inch video cassette; however, 1/2 inch VHS and Beta formats are also available on request.

2. Non-DoD organizations and the general public can obtain tapes at minimal cost, in any of the formats specified above, by writing to:

National Audio Visual Center
GSA
ATTN: Order Section
Washington, DC 20409

3. For additional information, contact:

Ada Joint Program Office
3D139 (400 A/N)
The Pentagon
Washington, DC 20301
(202) 694-0209

or:

Director
US Army Materiel Systems Analysis Activity
ATTN: DRXSY-MP (Herbert E. Cohen)
Aberdeen Proving Ground, MD 21005-5071
(301) 278-6577/6597

TABLE OF CONTENTS

Next page intentionally left blank.

READING ASSIGNMENTS

TAPE #1	"Programming in Ada" by J. G. P. Barnes (1982), Addison - Wesley Publishers Limited, pages 1-62.
TAPE #2	Data General, "Ada Work Center", pages 35-42 of text under Tape #2.
TAPE #3	"Programming in Ada", by J. G. P. Barnes, Chapter 6. Military Standard, ANSI/MIL-STD-1815A, dtd 22 Jan 83, "Ada Programming Language", pages 1-1 to 3-16.
TAPE #4	Review Chapter 5 - "Programming in Ada" by Barnes. Chapter 7 - "Programming in Ada" by Barnes. Chapter 10 - "Programming in Ada" by Barnes. Chapter 5 - Military Standard, ANSI/MIL-STD-1815A
TAPE #5	Review Chapter 7 - "Programming in Ada" by Barnes. Chapters 8 & 9 - "Programming in Ada" by Barnes
TAPE #6	Chapter 11 - "Programming in Ada" by Barnes Review Chapters 4 & 6 - "Programming in Ada" by Barnes Chapters 3 & 4 - Military Standard, ANSI/MIL-STD-1815A
TAPE #7	Review Chapters 7,8,9,11 - "Programming in Ada" by Barnes Chapters 6,7 - Military Standard, ANSI/MIL-STD-1815A Read Section 16.5 - "Programming in Ada" by Barnes
TAPE #8	Review Chapters 7.6 - "Programming in Ada"by Barnes Chapter 8 - Military Standard, ANSI/MIL-STD-1815A
TAPE #9	Sections 15.1 and 15.2 - "Programming in Ada" by Barnes Military Standard, ANSI/MIL-STD-1815A, Chapter 14
TAPE #10	Chapter 12 - "Programming in Ada" by Barnes Military Standard, ANSI/MIL-STD-1815A, 3.5.6 to 3.5.10 4.5.7, 4.6 and Annex A
TAPE #11 & #12	Chapters 13, 11.3 to 11.5 - "Programming in Ada" by Barnes Chapter 12 and 3.8 - Military Standard, ANSI/MIL-STD-1815A
TAPE #13	Chapter 14 - "Programming in Ada" by Barnes Military Standard, ANSI/MIL-STD-1815A, Chapter 9
TAPE #14	Chapters 15 & 16 - "Programming in Ada" by Barnes Chapter 13 - Military Standard, ANSI/MIL-STD-1815A
TAPE #15	No reading assignment.

TAPE #1

INTRODUCTION/GETTING STARTED

- DEFINING COMPONENTS
- GENERALIZING COMPONENTS
- DESIGN GUIDELINES
- SCOPE AND VISIBILITY
- RECORD ABSTRACTION
- NUMERIC ABSTRACTION
- DERIVED TYPES

```
pragma MAIN:
with TEXT_IO; use TEXT_IO; with SORT; use SORT:
procedure Roots is
  A, B, C: FLOAT;
  D: Float;
begin



--COLLECT PARAMETERS
PUT ("to solve AX**2+BX+C=0");
  NEW_LINE;
PUT ("A="); GET (A);
PUT ("B="); GET (B);
PUT ("C="); GET (C);
NEW_LINE (2);
--COMPUTE DISCRIMINANT
```

```
pragma MAIN;
with TEXT_IO; use TEXT_IO;
with SORT;
procedure ROOTS is
    --ax^2 + bx + c = 0
    A, B, C: FLOAT;
    D: FLOAT; --DISCRIMINANT
begin
```

EXAMPLE #2

WRITE A PROGRAM TO COMPUTE
THE ROOTS OF A QUADRATIC
EQUATION

```
pragma MAIN;
with TEXT_IO; use TEXT_IO;
procedure ADD is
   A,B,C: FLOAT;
begin
   GET (A); GET (B); GET (C);
   PUT ("SUM="); PUT (A+B+C);
end ADD;
```

```
pragma MAIN;
with TEXT_IO; use TEXT_IO;
procedure ADD is
   A,B,C: INTEGER;
begin
   GET(A); GET(B); GET(C);
   PUT ("SUM="); PUT (A+B+C);
end ADD;
```

GETTING STARTED

Ada® STANDARD

REFERENCE MANUAL pub. July 1980

MIL-STD 1815 Designated Dec. 1980

ANSI Canvass initiated Apr. 1980

ANSI Recanvass initiated Oct. 1980

ANSI Recanvass completed Sept. 1982

ANSI/MIL - STD 1815 Ada Jan 1983

Ada® LANGUAGE SPECIFICATION

REQUEST FOR PROPOSAL (APR 77)
17 LANGUAGE PROPOSALS RECIEVED

PHASE 1 (AUG 77 - FEB 78)

SOFTECH INTERMETRICS
SRI HONEYWELL

PHASE 2 (APR 78 - APR 79)
INTERMETRICS HONEYWELL

PHASE 3 (MAY 79 - JULY 80)
HONEYWELL

Ada REQUIREMENTS

- o STRONG TYPING
- o ENCAPSULATED DEFINITION
- o COMPOSITE TYPES
- o GENERIC DEFINITIONS
- o NUMERIC PRECISION
- o PARALLEL PROCESSING
- o EXCEPTION HANDLING
- o DECLARATION OF MACHINE DEPENDENCE

ADA® REQUIREMENTS DEFINED IN A
SERIES OF DRAFT SPECIFICATIONS

- o STRAWMAN (1975)
- o WOODMAN (1975)
- o TINMAN (1976)
- o IRONMAN (1977)
- 0 STEELMAN (1978)

EMBEDDED COMPUTER SYSTEMS
SOFTWARE CHARACTERISTICS

- o LARGE
- o LONG LIVED
- o CONTINUOUS CHANGE

- EMBEDDED COMPUTER SYSTEMS APPLICATIONS CHARACTERISTICS
- REATL TIME CONSTRAINTS
- AUTOMATIC ERROR RECOVERY
- CONCURRENT CONTROL
- NON-STANDARD INPUT-OUTPUT

THE MOTIVATION FOR

Ada

- GENERICS
- TASKING
- ACCESS TYPES
- TASK TYPES
- MACHINE DEPENDENT CODE

```
PUT ("C="); GET (c);
NEW_LINE (2);


--COMPUTE DISCRIMINANT
D:=B**2 - A.0*A*C;
--REAL ROOTS?
if D >= 0
   then
     PUT ("POSITIVE ROOT =");
     PUT ((-B + SQRT(D)) / (2.0*A)
     NEW_LINE;
     PUT ("NEGATIVE ROOT =");
     PUT ((-B - SQRT(D)) / (2.0*A))
     NEW_LINE;
   else
     PUT ("IMAGINARY ROOTS");
   end if;
end ROOTS;
```

```
if D >= 0
        then
          PUT ("POSITIVE ROOT = ");
     PUT ((-B + SORT(D))/(2.0*A));
          NEW_LINE;
          PUT ('NEGATIVE ROOT =");
          PUT ((-B - SORT(D))/(2.0*A));
          NEW_LINE;
        else
          PUT ("IMAGINARY ROOTS");
     end if ;
```

MAIN PROCEDURE TEMPLATE

```
pragma MAIN;              --FOR ADE
with COMPONENTS;          --LIBRARY
use COMPONENTS;
procedure NAME is
  {DECLARATIVE_PART}
begin
  {SEQUENCE_OF_STATEMENTS}
end NAME;
```

BUILT-IN TYPES

Variables	Literals
I: INTEGER;	2 or 3 or 789_123
A: FLOAT;	2.0 or 1.0E35
S: STRING (1..5);	"HELLO"

TYPE CONVERSION

FOR "CLOSELY RELATED" TYPES

o I:= INTEGER (A); --WILL ROUND

o A:= FLOAT (I);

o STRING:="123_456"--CHARACTER STRING

o I:= ~~INTEGER (STRING)~~ --ILLEGAL

ASSIGNMENT STATEMENT

```
VARIABLE := EXPRESSION;
```

EXPRESSIONS

o OPERATOR PRECEDENCE

+ -

* / mod rem

**

o LEFT TO RIGHT

o EXAMPLES

```
B**2 - 4.0*A*C
(-B+SQRT(D)) / 2.0*A
```

```
package TEXT_IO
GET (A); --ASSUMES FLOAT_IO (FLOAT)
PUT (A);
GET (I); --ASSUMES INTEGER_IO (INTEGER)
PUT (I);
GET (STRING); --VARIABLE LENGTH
PUT ("HELLO");
```

OVERLOADING

THE SAME IDENTIFIER HAS MORE THAN ONE MEANING AT A GIVEN POINT IN PROGRAM TEXT

```
pragma MAIN;
with TEXT_IO; use TEXT_IO;
procedure ADD is
  A,B,C : FLOAT;
begin
  GET (A); GET (B); GET (C);
  PUT ("SUM ="); PUT (A+B+C);
end ADD;
```

LEXICAL ELEMENTS

(CHAPTER 2)

- IDENTIFIER ::= letter

  {[underline] letter_or_digit}

  --USED AS NAMES AND RESERVED

  --WORDS

- DELIMITERS ::=

  & ' () * + , - . / : ; < = > |

  => .. ** := /= >= <= << >> <>

- SEPARATOR ::= SPACE |

  FORMAT_EFFECTOR | EOL

COMPOUND DELIMITERS

=>	ARROW
..	DOUBLE DOT
**	DOUBLE STAR
:_	BECOMES
/=	NOT EQUAL
>=	GT OR EQ
<=	LT OR EQ
<<	LEFT LABEL BRACKET
>>	RIGHT LABEL BRACKET
<>	BOX

if STATEMENT

```
IF_STATEMENT ::=
   if CONDITION then
     SEQUENCE_OF_STATEMENTS
           . . .
   [else
     SEQUENCE_OF_STATEMENTS]
   end if;
```

IF STATEMENT

(EXAMPLE)

```
if (COLD and SUNNY) or WARM
 and then STATE = "VA"
  and MONTH in WINTER
 then
 . . .
 end if;
```

MV/FAMILY OVERVIEW

ECLIPSE MV Family of Systems incorporate an advanced 32-bit architecture with up to 16 Megabytes of physical main memory. Efficient demand paging techniques, cache structures, and instruction pipelining let the system make use of its 4 Gigabyte logical address space with maximum efficiency. Individual program user space can be as large as 2 Gigabytes. This gives the system the high capacity and performance needed to support multi-user Ada program development as well as real-time, multiprogramming Ada applications.

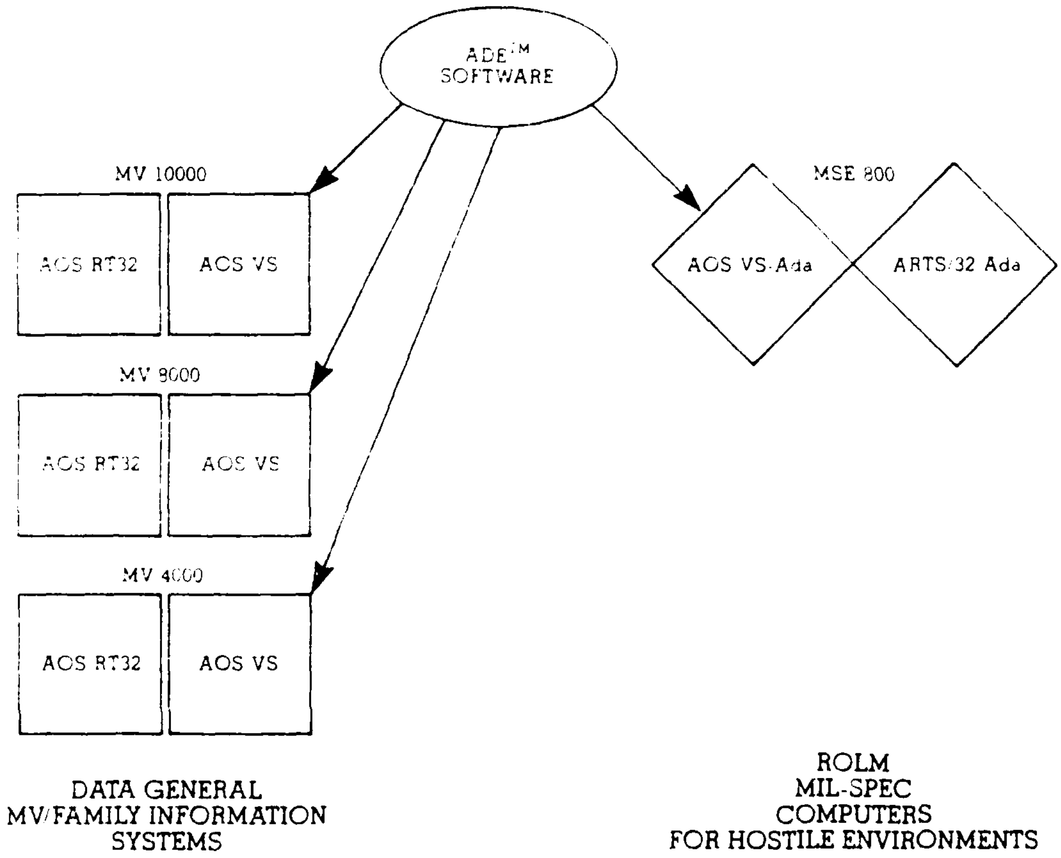
ECLIPSE MV Family systems feature security mechanisms which complement the object orientation of Ada. The system's 4 Gigabyte virtual address space is divided into 8 processing segments of 512 Megabytes each for efficient memory management. These processing segments are identical to the hierarchical ring structure that is used to protect system resources.

These systems are designed for Availability, Reliability, and Maintainability. The System Control Processor (SCP) performs self-diagnostics on internal subsystems, maintains a log of system errors, and identifies hardware faults to the field-replaceable unit level. It lets operators inspect memory and step through programs, in order to monitor both hardware and software performance. In addition, the mechanical design of MV/Family systems facilitates accessibility and efficient maintenance.

MV/FAMILY HARDWARE OVERVIEW

	MV/4000	MV/8000	MV/10000
Maximum Address Space	4GB	4GB	4GB
Maximum User Program Size	2GB	2GB	2GB
Whetstone Performance (DP/FPU)	400	995	1900
On-Line Storage	4.7GB	9.6GB	18.5GB
Floating Point Unit	opt	opt	std
Maximum Main Memory	8MB	12MB	16MB
System Cache	N/A	16KB	16KB
Instruction Cache	N/A	std	std

DEPLOYMENT STRATEGIES



Procedures—Often a pro-
r has an operation to per-
 example, sorting an array),
logic does not depend on the
data it is operating with. By
a generic procedure in Ada
 programmer may specify
at types of data it is to oper-
The logic and the code need
veloped only once. Having
of such generic proce-
lps increase programmer
vity.

VELOPMENT
NMENT

a language and its associated
rs are only one aspect of the
rk Center. The entire pro-
nd environment supports the
ment effort. The Ada Devel-
Environment is designed to
ent the guidelines and the
the DoD STONEMAN speci-
Included are capabilities
ment preparation, appli-
 program development and
project management (in-
configuration control and
g) and system management.
ment tools, and the required
se, are built as a logical layer
 AOS/VS operating system.
ical layer is referred to as the
Ada Programming Support
ment. The "KAPSE" shields
ication programmer from
ils of the host operating sys-
 significantly reduces pro-
er training requirements. In
 is the DoD solidifies the
vironment specification,
APSE" will allow adoption
e standards for the Ada
Center.

User Interface-Command Line Interpreter (CLI)

The CLI is the primary interface between the programmer and the Ada Development Environment. The CLI controls terminal sessions, access to tools, and provides a user "help" facility.

Data Base Control Tools

The Ada Development Environment has four data control tools: the Data Base Manager (DBM), consisting of the Configuration Control Manager (CCM), the Mapper, and the Librarian. The DBM predefines data base primitives and allows definition of user primitives. It also provides services for creating, accessing, modifying, relating and deleting all ADE data base objects. The CCM provides control over the manipulation of ADE data base objects, including archiving and revision control services. The Mapper provides the means by which library objects can be specified and located. Finally, the Librarian is responsible for controlling the logical groupings of objects comprising Ada library units and subunits, as well as controlling access to those objects.

Application Development Tools

These tools include the Editor, Formatter, Pretty Printer, File Maintainer, and Debugger. The Editor is used by programmers to enter Ada source text, as well as other textual materials; it is capable of Ada-indenting and format control. The Formatter processes text files and reformats them into documentation files. The Pretty Printer is responsible for printing Ada programs in a logical Ada format. The File Maintainer allows comparisons of object programs; text files and typeless files can each be compared. The Debugger provides a symbolic debugging facility to aid in the testing of Ada application programs.

Target Development Tools

Several of the tools are configured to support specific target machines. These tools include the Ada compilers themselves, Runtime Support Packages, Assemblers, Object Importers, Linkers, and Exporters. Ada compilers with unique code generators will be available for each of the target CPU's.

Unique Runtime Support Packages are supplied for each of the target environments. Each target also requires its own Assembler, which will be available as a cross-development tool. The Object Importer is used to bring into the Ada Development Environment binary modules produced by other language compilers such as FORTRAN 77. The Linker combines Ada-binary with Libraries and Runtime Support Packages to create Ada Program Files. The Exporter tools are responsible for formatting and transferring Ada Program Files from the host environment to the target environments.

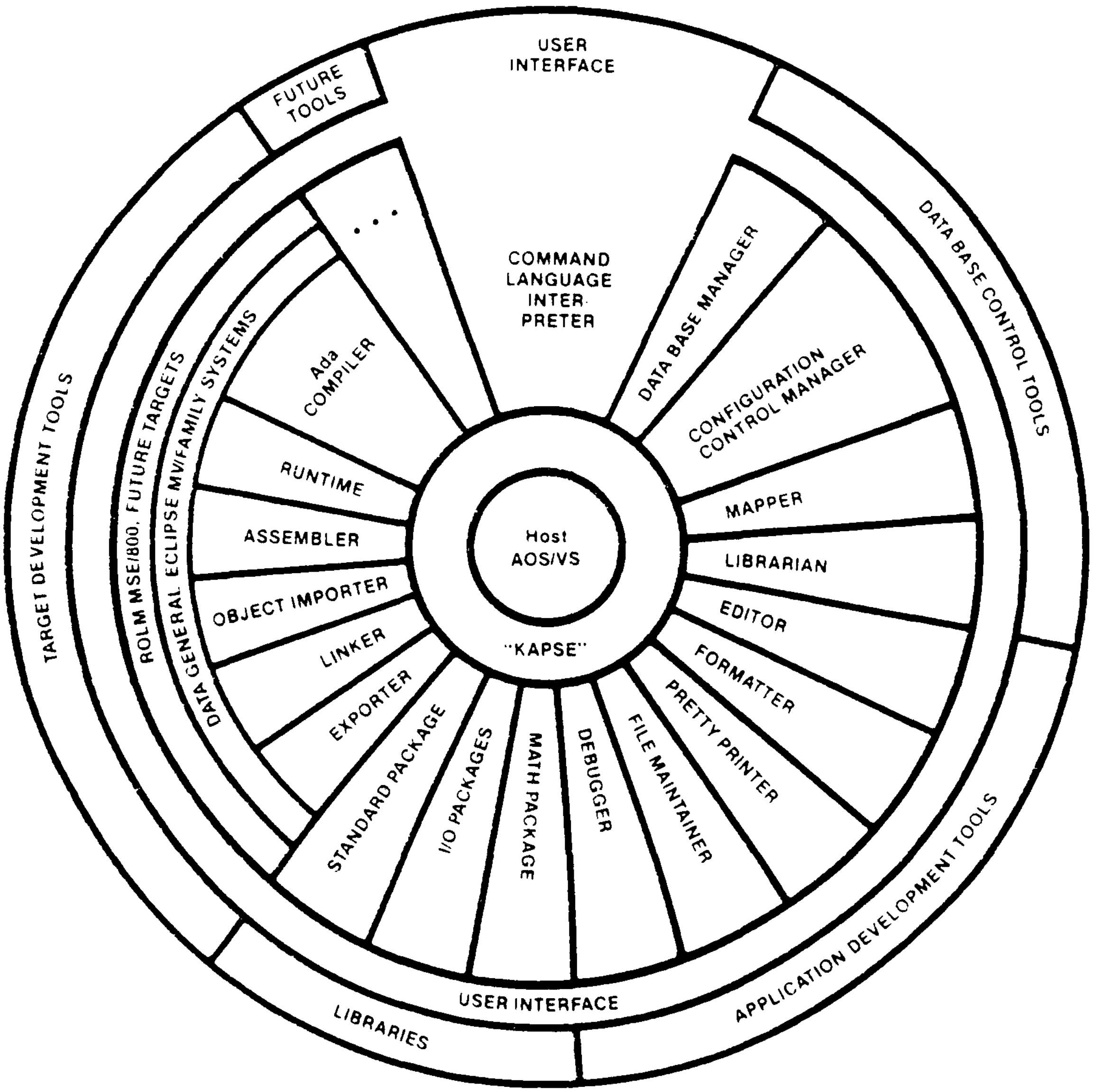
Libraries

The Ada Work Center includes Libraries of Packages that are useful to application programs. The STANDARD Package includes all of the basic language definitions, such as data types, allowable operations, and predefined exceptions. I/O Packages define fundamental data input/output capabilities such as sequential and direct I/O. A Math Package provides users with programs for computing commonly used math functions, such as sine and cosine.

SOFTWARE OPTIONS

The Ada Work Center supports non-Ada language program development concurrently with Ada program development. Any software available under the AOS/VS operating system is available on the Ada Work Center.

ADE™ SOFTWARE



ADE-Ada Development Environment features full DOD-Spec. ANSI Standard Ada Compiler and the most complete set of integrated Ada environment tools available.

programs developed on any of the Ada Work Center systems can [illegible] for execution on 32-bit [illegible] Family and 32-bit [illegible] MSE 800 systems.

[illegible] Ada Work Center is a fully in[illegible] Ada program development [illegible] System hardware, software [illegible] support, and full software [illegible] and support are all provided [illegible] Data General. The Ada Work [illegible] is designed for the user who [illegible] the full implementation [illegible] Ada language and wants to [illegible] immediately productive [illegible]

[illegible] PROGRAMMING LANGUAGE [illegible] LANGUAGE FEATURES

[illegible] Ada compiler in the Ada Work [illegible] is an implementation of the [illegible] DoD and ANSI standards. A [illegible] description of the features [illegible] is presented in the 1983 Ada [illegible] Reference Manual. Some [illegible] features that contribute to the [illegible] of Ada are described here:

[illegible]dern Software Engineering [illegible]thodologies—The syntactic struc[illegible] of the Ada language strongly [illegible]ports orderly program develop[illegible]. As a block-structured [illegible]guage, Ada facilitates logical [illegible]king and orderly coding. Top-[illegible] methodologies are supported [illegible]parate compilation capabili[illegible] and by the separation of the [illegible]fication and implementation [illegible] of an object. Bottom up metho[illegible]ogies are supported by the use of [illegible]aries of packages. Because of [illegible] support for these methodologies, [illegible] is used by some organizations [illegible] systems design specifi[illegible]tion tool.

Object Orientation—Ada introduced the concept of packages. A package consists of a specification part and an implementation part, which can be compiled separately. References to a package are only made to the specification part. This concept also supports information hiding, i.e., the formats of data structures need not be known to other packages and sub-programs which use that data. Data can only be accessed through the procedures directly controlling the data.

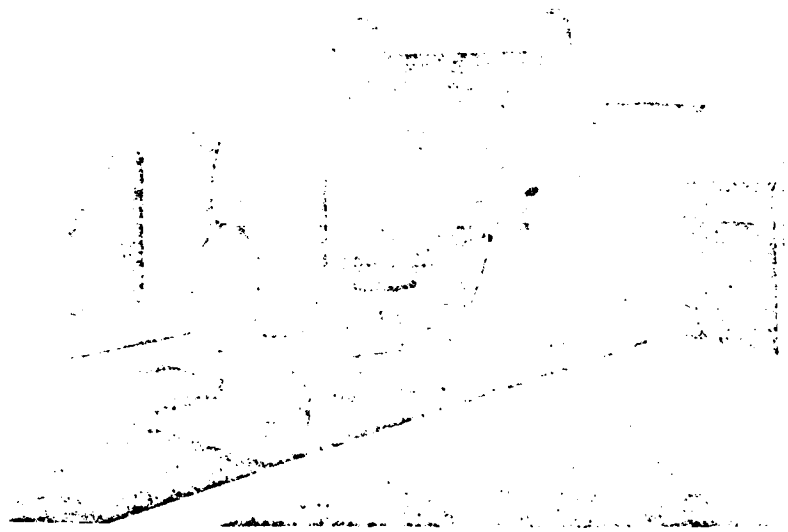
Another aspect of the Ada object orientation is a comprehensive separate compilation facility. This allows subordinate packages, sub-programs and tasks to be separately compiled as subunits. Although the subunits are kept as separate data base objects, their specification parts remain in the parent packages and subprograms. Thus, the entire executable program can easily be constructed at link time. This object orientation of the Ada language promotes the attainment of the software reliability, portability, and reuse-ability goals.

Strong Data Typing—All data must be typed. There is a rich assortment of natural data types which are defined in the Ada language, and users may define their own types. Creation of user data types not only enhances the self-documenting features of Ada, but it facilitates logical correctness. Only similar data types may be combined in logical or arithmetic operations. The compiler is responsible for checking the correctness of all operations at compile-time and at runtime.

Complex Data Structures—The compiler allows the user to create complex data structures. Arrays of arbitrary dimensions and dynamic size can be created. Strings are created as arrays of characters. Records—collections of data of dissimilar types—are supported. Variant records, where the format varies as a function of values of data within individual records, can be created. Data structures can be composed in arbitrarily complex ways allowing arrays of records, records embedded within records, arrays within records, and so on.

Concurrency and Multitasking—The language definition of Ada supports concurrent operations. This concurrency is based on a tasking model, which includes intertask communications and synchronization protocols, and parallel execution. This is an important feature for real-time applications.

Exception Handling—Ada has the ability to recover from execution time errors such as arithmetic faults, hardware faults, and array bounds errors. Further, the user may specify at what level in the program this error should be handled. Errors are handled where they logically should be handled, rather than allowing their effects to propagate destructively, or allowing the operating system to deal with them in an arbitrary way. This gives the user a great deal of control over real-time situations.



FEATURES

- Fully integrated hardware and software Ada® Work Center.
- Also available as full Ada software development system.
- Accepts full Ada language as defined in the 1983 Ada Language Reference Manual, MIL-STD-1815(A) and ANSI MIL-STD 1815A-1983.
- Extensively tested against the available DoD ACVC Validation Suite.
- Compilers, code generators, and major environment tools written in Ada.
- Hosted on powerful, 32-bit ECLIPSE® MV/Family Systems.
- Produces executable Ada object programs which can be targeted at 32-bit ECLIPSE MV/Family and ROLM® Mil-Spec computer systems.
- Host Target hardware and software I/O compatibility.
- Ada Development Environment (ADE™ software) using guidelines of the DoD STONEMAN Specification is built on AOS/VS Operating System.
- AOS/VS FORTRAN 77 object code can be integrated and supported in the Ada Development Environment.
- Supports from 5 to 15 interactive Ada program development workstations.

The Ada Work Center is a complete configuration that includes hardware, software, and support and allows users to become immediately productive in the development of applications in the Ada language. The hardware portion of the system centers on the high performance, 32-bit ECLIPSE MV/Family Systems. The Ada compiler accepts the full Ada language as defined by MIL-STD-1815(A) and by the ANSI MIL-STD 1815A-1983.

The Ada Work Center also includes a comprehensive Ada programming environment to support development and maintenance activities. Based upon guidelines described in the DoD STONEMAN specification taking full advantage of the proven power and versatility of the Data General AOS/VS virtual memory operating system, the Ada Development Environment will substantially increase programmer productivity and minimize costs associated with program development, testing, and maintenance.

ECLIPSE MV/Family Systems provide ideal hosts for the Ada compiler and Ada Development Environment. Four packaged Ada Work Center Systems are available: Two ECLIPSE MV/4000® based systems are excellent entry level systems for up to 5 concurrent Ada developers. For up to 8 users, an ECLIPSE MV/8000® based Ada Work Center is a cost effective choice. The ECLIPSE MV/10000™ based Ada Work Center is the most powerful configuration and supports up to 15 simultaneous users. Each Ada Work Center comes complete with a system console, disk storage, a magnetic tape unit and optional line printer. The system may be expanded by the addition of more peripherals, memory, and terminals. In addition, the full line of ECLIPSE hardware and software options is available.

READING ASSIGNMENT - TAPE #2

DATA GENERAL - ADA WORK CENTER

NEXT PAGE INTENTIONALLY LFET BLANK.

TAPE #2

RUNNING A PROGRAM

NEXT PAGE INTENTIONALLY
LEFT BLANK.

REVIEW

- o ASSIGNMENT
- o I/O - GET, PUT, NEW_LINE
- o FLOW OF CONTROL - IF, LOOP
- o DECLARATIONS - TYPE AND OBJECT
- o EXPRESSIONS - ARITHMETIC AND LOGICAL
- o TYPE CONVERSIONS
- o MAIN PROCEDURE

ELABORATION OF STRING

DECLARATIONS

S: STRING (1..5);

~~T: STRING := "HELLO~~"; --ILLEGAL

T: constant STRING := "HELLO";

--STORAGE, AND HENCE SIZE,

--FIXED WHEN DECLARATION

--IS ELABORATED

ELABORATION

DECLARATIONS INVOLVE
RUN-TIME ACTIVITY :

1) STORAGE ALLOCATION -
   CREATE OBJECTS
2) INITIALIZE OBJECTS

EXAMPLE -

```
I : INTEGER := 0;
```

```
for COLOR in COLOR_CHART
  loop
    --TRY COLOR
  exit when GOOD;
end loop;
```

LOOP STATEMENT

```
LOOP_STATEMENT ::=
  [LOOP_SIMPLE_NAME:]
  [INTERATION_SCHEME] loop
   SEQUENCE_OF_STATEMENTS
end loop [LOOP_SIMPLE_NAME];
INTERATION_SCHEME ::=
  while CONDITION |
  for LOOP_PARAMETER_SPECIFICATION
```

LOGICAL EXPRESSIONS

- <u>and</u> | <u>or</u> | <u>xor</u>
  - --ALLOW ARGUMENTS TO BE
  - --EVALUATED IN EITHER ORDER
- <u>and then</u> | <u>or else</u>
  - --LEFT ARGUMENT FIRST
- RELATIONAL OPERATORS PLUS <u>in</u>, <u>not in</u>

MV FAMILY SOFTWARE OVERVIEW

Data General offers a broad range of [illegible] software for ECLIPSE MV Family Systems. The AOS/VS [illegible] memory operating system [illegible] multiprogramming and [illegible] capabilities with [illegible] AOS/VS [illegible] program development and [illegible] for both 16 and 32 bit [illegible] AOS/VS is also compatible with Data General's [illegible] software [illegible] program development, [illegible] data management and [illegible].

[illegible] Data General [illegible] Data General [illegible] PRESENT [illegible] TRENDVIEW [illegible] the CEO Document [illegible] CEO and [illegible] word processing systems [illegible].

For program development, Data General offers ANSI and industry [illegible] as well as [illegible] ANSI [illegible] BASIC [illegible] FORTRAN [illegible].

[illegible] for the PROXI [illegible] BOL Program Generator [illegible] and the MPL Macro processor [illegible] languages.

For data management, Data General offers DG/DBMS [illegible] database management software, INFOS II file management software and the SORT/MERGE utility.

For communications networking, Data General offers [illegible] X.25 [illegible] Data General [illegible] SNA for IBM environments.

DEPLOYMENT STRATEGIES

Data General [illegible] ECLIPSE MV [illegible] AOS/VS operating system [illegible] requirements [illegible] the Ada Work Center is [illegible] compatible with [illegible].

Ada WORK CENTER PACKAGED CONFIGURATIONS

Model [illegible]	[illegible] A	[illegible] B	[illegible]	[illegible]
Host Processor	MV/4[illegible]	MV/4[illegible]	MV/[illegible]	MV/[illegible]
Main Memory (MB)	[illegible]	4	[illegible]	[illegible]
On-Line Storage (MB)	[illegible]	[illegible]	[illegible]	[illegible]
Magnetic Tape Units (bpi)	[illegible]	[illegible]	[illegible]	[illegible]
Floating Point Unit	yes	yes	yes	[illegible]
System Console	D4[illegible]	D4[illegible]	TP[illegible]	TP[illegible]
Battery Backup	yes	yes	yes	yes
D410 Terminal Ports	[illegible]	[illegible]	[illegible]	24
Software:				
AOS/VS Operating System	yes	yes	yes	[illegible]
Ada Development Environment	yes	yes	yes	[illegible]
Software Support	[illegible]	[illegible]	[illegible]	[illegible]

To each packaged Ada Work Center, Data General offers a wide variety [illegible] Work Center [illegible] the [illegible] environment.

NORTH AMERICAN OFFICES Westboro Massachusetts 01580 (617) 366-8911 (head quarters) And AL Birmingham Montgomery AR Little Rock AZ Phoenix Tucson CA El Segundo Fresno Los Angeles Oakland Palo Alto Pasadena Riverside Sacramento San Diego San Francisco Santa Ana Santa Barbara Van Nuys CO Denver Colorado Springs Ft. Collins CT [illegible] Milford North Branford FL [illegible] [illegible] Jacksonville Orlando Tampa GA [illegible] HI Honolulu IA Bettendorf W. Des Moines ID Boise IL Arlington Heights Champaign Chicago Rockford Peoria IN Indianapolis KY Louisville LA Baton Rouge Metairie MA Wellesley West Springfield Worcester MD Baltimore ME Portland MI Birmingham Grand Rapids MN Minnetonka MO Kansas City St. Louis MS Jackson MT Billings NC Charlotte [illegible] Greensboro Greenville Raleigh NE Omaha NH Bedford NJ Somerset Wayne NM Albuquerque NV Reno NY Amherst [illegible] Lake [illegible] Latham Liverpool Melville New York City Rochester White Plains OH Cincinnati Columbus Dayton Independence OK [illegible] Tulsa OR Lake Oswego PA [illegible] Philadelphia Pittsburgh RI Providence SC Columbia Greenville TN [illegible] Memphis Nashville TX Austin Dallas Ft. Worth Houston San Antonio UT Salt Lake City VA McLean Norfolk Richmond [illegible] WA Bellevue Richland WI Appleton [illegible] Madison WV St. Albans **CANADA** ALBERTA Calgary Edmonton BRITISH COLUMBIA Vancouver MANITOBA Winnipeg NOVA SCOTIA Halifax ONTARIO Mississauga Ottawa Toronto QUEBEC Montreal

INTERNATIONAL OFFICES AUSTRALIA Adelaide Brisbane Canberra Darwin Hobart Melbourne New Castle Perth Sydney AUSTRIA Vienna BELGIUM Brussels BRAZIL Sao Paulo CHILE Santiago DENMARK Glostrup FRANCE **PARIS EUROPEAN HEADQUARTERS** Les Plessis Robinson Lille Lyon Nantes Orsay Saint Denis Strasbourg HONG KONG IRELAND Dublin Belfast ITALY Milan Rome Padua Turin Florence Bologna JAPAN Gyoda Tokyo Sapporo Sendai Tsukuba Nagoya Osaka Seibo Hiroshima Fukuoka NETHERLANDS Amsterdam Rijswijk Zwolle Badhoevedorp NEW GUINEA Papua NEW ZEALAND Wellington Auckland Christchurch PUERTO RICO Guaynabo SCOTLAND Glasgow SINGAPORE SPAIN Madrid Barcelona SWEDEN Stockholm Malmo Gothenburg SWITZERLAND Zurich Lausanne Basle Bern THAILAND Bangkok TRINIDAD Port of Spain UNITED KINGDOM Birmingham Bristol Hammersmith Hounslow Leeds London Manchester Northolt South Harrow Warrington VENEZUELA Caracas FEDERAL REPUBLIC OF GERMANY Frankfurt Hamburg Hannover Dusseldorf Stuttgart Nuremberg Munich

REPRESENTATIVES/DISTRIBUTORS ARGENTINA Buenos Aires BOLIVIA Santa Cruz De La Sierra COLUMBIA Bogota COSTA RICA San Jose ECUADOR Quito EGYPT Cairo FINLAND Espoo GREECE Athens GUATEMALA Guatemala City INDIA Bombay INDONESIA Jakarta ISRAEL Givatayim IVORY COAST Abidjan JORDAN Amman KOREA Seoul KUWAIT Safat LEBANON Mkalles MALAYSIA Kuala Lumpur MEXICO Mexico City Garza Garcia NIGERIA Lagos Ibadan Kaduna NORWAY North Oslo PARAGUAY Asuncion PERU Lima PHILIPPINES Manila PORTUGUAL Amadora SAUDI ARABIA Riyadh Jiddah SOUTH AFRICA Durban Pretoria Johannesburg Cape Town SRI LANKA Colombo TUNISIA Tunis TURKEY Ankara UNITED ARAB EMIRATES Abu Dhabi ZIMBABWE Harare

The materials contained herein are summary in nature, subject to change, and intended for general information only. Details and specifications concerning the use and operation of Data General equipment and software are available in the applicable technical manuals available through local sales representatives.

Data General
A Generation Ahead.

Ada

DEVELOPMENT ENVIRONMENT

$\boxed{\text{ADE HOST}}$	DEBUGGING TOOLS	$\boxed{\text{ADE TARGET}}$
LIBRARIAN		ISA
SOURCE EDITOR		RUN-TIME
COMPILER		LOADER
LINKER		DEVICES
. . .		. . .

LOGGING ON

< CR >

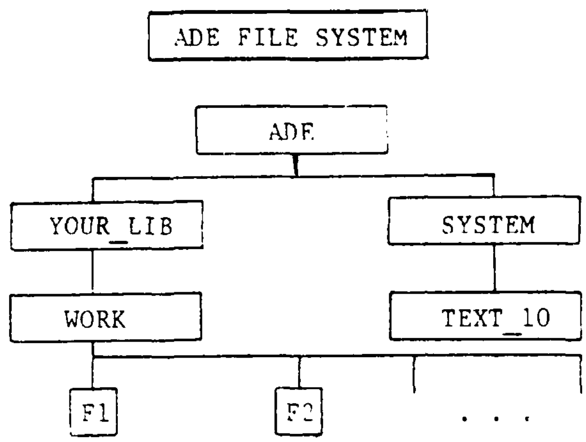
USERNAME: bill <CR>

PASSWORD: x <CR>

AOS CLI

) aenter

=>


ADE FILE SYSTEM
ADE
YOUR_LIB
SYSTEM
WORK
TEXT_10
F1
F2
. . .

AEDIT

INSERT	2
MODIFY	4
DELETE	4 6
MOVE	3 5 BEFORE 7
DUP	3 5 BEFORE 7
SCREEN AHEAD	Func 2
SCREEN BACK	Func 1
BYE	

FILE NAMES

```
OBJECT NAME::=
   IDENTIFIER (1..30) / {QUALIFIERS}
QUALIFIERS:
   CATEGORY   (ADA, BIN, PROG, LIST)
   VERSION    1.0, 2.0, . . .
   TARGET     ADE, AOS_VS, . . .
```

COMMAND EXAMPLES

ALIB	ADE: YOUR_LIB:WORK
ACREATE	FIRST/CAT=ADA
AEDIT	FIRST
ACOMP	FIRST/TAR=ADE
ATYPE	FIRST/CAT=LIST
ALINK	FIRST
AEXEC	FIRST

STATEMENTS DISCUSSED THUS FAR

- o DECLARATIONS
- o GET, PUT, NEW_LINE
- o ARITHMETIC EXPRESSIONS
- o ASSIGNMENT
- o MAIN PROCEDURES
- o IF...THEN...ELSE
- o TYPE CONVERSION

LABORATORY #1

LABORATORY #1

```
--READ AN INTEGER (N)
--READ N INTEGERS
--ADD THEM
--PRINT RESULT
--MODIFY YOUR PROGRAM TO ADD
--   N REAL NUMBERS
--EXPERIMENT WITH GET AND PUT
--FOR OTHER TYPES
```

OBJECTIVES OF LAB #1

1) LEARN TO USE ADE
2) COMPILE, LINK, AND EXECUTE
   AN ADA PROGRAM
3) EXPERIMENT WITH LITERALS
4) BUILD FAMILIARITY WITH TEXT_IO
   (SEE SECTION 14.3)

1) PROGRAM NAME
   SAME AS FILE NAME

2) GET AND PUT
   OPTION 1:
   with TTY_IO
   use TTY_IO

   OPTION 2:
   instantiate TEXT_IO
   for INTEGER
   &/OR FLOAT

INSTANTIATE FLOAT_IO

```
pragma MAIN
with TEXT_IO
 procedure TEST 1 is
   package REAL_IO
     is new
     TEXT_IO. FLOAT_IO
           (FLOAT);
 use REAL_IO;
```

GENERICS

- ENUMERATION_IO is generic
  A new version must be
  instantiated for each
  enumeration type

ENUMERATION I/O

```
--SEE 14.3
with TEXT_IO; use TEXT_IO;
package DAY-IO is new
   ENUMERATION_IO (DAY);
IN_DAY: DAY;
GET (IN_DAY);
PUT ("DAY=");
PUT (IN_DAY);
```

ENUMERATION TYPES

```
type DAY is
    (MON,
    TUES,
    WED,
    THURS,
    FRI,
    SAT,
    SUN);
```

INDEX VALUES

- DISCRETE TYPE

  INTEGER

  or

  ENUMERATION

- DISCRETE RANGE

  CLOSED INTERVAL OF

  VALUES OF A

  DISCRETE TYPE

UNCONSTRAINED ARRAY

```
subtype POSITIVE is INTEGER
   range 1..INTEGER'LAST;
type STRING is array
   (POSITIVE range <>)
   of CHARACTER;
V:STRING (1..5);
```

MULTIDIMENTIONAL ARRAYS

```
RECTANGLE: array
   ( 1..20, 1..30 )
   of FLOAT;
tvpe SCHEDULE is
   arrav ( WEEK,   --1..52
           DAY,    --MON..SUN
           HOUR)   --1..24
   of STRING;
```

ONE DIMENSIONAL ARRAYS

```
type VECTOR is
   array (1..10)
   of INTEGER;
type LINE is
   array (1..MAX_LINE_SIZE)
   of CHARACTER;
type SCHEDULE is array (DAY)
   of BOOLEAN;
```

RECORDS

```
type TIME is
   record
      DAY : INTEGER
        range 0..366*(2099-1901+1);
      SECOND : DURATION  --see 9.6
        range 0.0..86_400.0;--one day
   end record;
```

TAPE #3

RECORDS, ARRAYS & ENUMERATION TYPES

NEXT PAGE INTENTIONALLY
LEFT BLANK.

THIS PAGE BLANK

TTY_IO

Is package in ADE Library

provides:

INTEGER_IO

CHARACTER_IO

STRING_IO

to terminal

use TTY_IO

```
                         with
        ______________________________
           |                    |
        TEXT_IO              TEST 1

        |-FLOAT_IO
        |-INTEGER_IO
        |-STRING_IO

          TEXT_IO.FLOAT_IO
           use TEXT_IO
```

INSTANTIATE FLOAT_IO

```
pragma MAIN
with TEXT_IO
procedure TEST 1 is
  package INT_IO
    is new
  TEXT_IO. INTEGER_IO
    for INTEGER
use INT_IO
```

SUBTYPES

```
subtype WEEKDAY is
   DAY range MON..FRI;
subtype WEEK is
   INTEGER range 1..52;
subtype HOUR is
   INTEGER range 1.. 24;
```

SUBTYPE

- o SUBSET OF VALUES OF BASE TYPE
- o DETERMINED BY A CONSTRAINT

OVERLOADING

```
type COLOR              type LIGHT
is (WHITE,              is (RED
   RED,                    AMBER,
   YELLOW,                 GREEN);
   GREEN);

   COLOR'GREEN /= LIGHT'GREEN
```

OVERLOADING

- THE SAME IDENTIFIER HAS MORE THAN ONE MEANING AT A GIVEN POINT IN PROGRAM TEXT

TYPE

- A SET OF VALUES

  and

- A SET OF OPERATIONS
  APPLICABLE TO
  THOSE VALUES

SUBTYPE

- SUBSET OF VALUES OF BASE TYPE
- DETERMINED BY A CONSTRAINT

OBJECT

o OBJECTS CONTAIN VALUES

o CREATED BY ELABORATING
  A DECLARATION (OR ...)

o TYPE BOUND AT
  ELABORATION

NEXT PAGE INTENTIONALLY
LEFT BLANK.

TAPE #4

FLOW OF CONTROL

NEXT PAGE INTENTIONALLY
LFET BLANK.

FLOW OF CONTROL STATEMENTS

1. SEQUENTIAL
2. IF. . .THEN. . .ELSE
3. CASE. . .IS. . .
4. WHILE. . .LOOP
5. FOR. . .LOOP
6. EXIT. . .WHEN. . .
7. PROCEDURE CALL
8. FUNCTION INVOCATION
9. RETURN

OTHER STATEMENTS AFFECTING
FLOW OF CONTROL

10. RAISE --EXCEPTION
11. TASK INITIATION
12. ENTRY CALL -
    RENDEZVOUS FROM USER TASK
    LOOKS LIKE A PROCEDURE CALL
13. ACCEPT -
    RENDEZVOUS FROM SERVER TASK
14. ABORT
15. ~~GO TO~~ -- DON'T USE

IF STATEMENT

```
if D >= 0
  then
    PUT ("POSITIVE ROOT =");
    --ETC.
  else
    PUT ("IMAGINARY ROOTS");
end if;
```

if STATEMENT

```
IF_STATEMENT ::=
   if CONDITION then
     SEQUENCE_OF_STATEMENTS
             . . .
   [else
     SEQUENCE_OF_STATEMENTS]
   end if;
```

ELSIF

SYNTACTIC CONVENIENCE

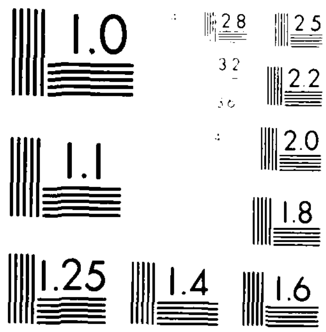
```
if CONDITION
  then . . .
  else if COND_2 then . . .
       else if COND_3 then . . .
            else
            end if; --COND_3
       end if; --COND_2
end if;
```

CASE STATEMENT

```
case SENSOR is
  when ELEVATION =>
    RECORD_ELEVATION
      (SENSOR_VALUE);
  when AZIMUTH =>
    RECORD_AZIMUTH
      (SENSOR_VALUE);
  when others =>null;
end case;
```

AD-A155 779 AN INTRODUCTION TO ADA (TRADEMARK) FOR SCIENTISTS AND ENGINEERS(U) ARMY MATERIEL SYSTEMS ANALYSIS ACTIVITY ABERDEEN PROVING GROUND MD H E COHEN OCT 83 2/5

UNCLASSIFIED F/G 9/2 NL


1.0
2.8
2.5
3.2
2.2
3.6
2.0
1.1
1.8
1.25
1.4
1.6

CASE STATEMENT

```
case DISCRETE_TYPE_EXPRESSION
  is --COVER VALUES ONCE
    when DISCRETE_RANGE  =>
     . . .
    when ANOTHER_DISCRETE =>
     . . .
    when other =>
     . . .
end case;
```

WHILE . . . LOOP

WHILE CONDITION EVALUATED
BEFORE EACH EXECUTION OF
THE SEQUENCE OF STATEMENTS

```
while CONDITION loop
   SEQUENCE_OF_STATEMENTS
end loop;
```

ONLY EXECUTES IF CONDITION TRUE

FOR . . . LOOP

<u>for</u> I <u>in</u> DISCRETE_RANGE

<u>loop</u> . . . <u>end</u> <u>loop</u>;

1. CREATES I
2. EVALUATES DISCRETE RANGE
3. IF DISCRETE RANGE NOT null
   EXECUTE, TREATING I AS CONSTANT
4. AFTER ALL VALUES, DESTROY
   I AND EXIT

LOOP STATEMENT

```
LOOP_STATEMENT  ::=
   [LOOP_SIMPLE_NAME:]
   [ITERATION_SCHEME]  loop
     SEQUENCE_OF_STATEMENTS
   end loop [LOOP_SIMPLE_NAME];
ITERATION_SCHEME  ::=
   while CONDITION
   | for LOOP_PARAMETER_SPEC
```

EXIT STATEMENT

```
OUTER_LOOP:
  for I in 1..10 loop
    for J
      in reverse 1..10 loop
      exit when J = I + 3;
      exit OUTER_LOOP
       when J = I;
    end loop;
  end loop OUTER_LOOP;
--I AND J NOT VISIBLE
```

EXIT STATEMENT

```
EXIT_STATEMENT  ::=
 exit [LOOP_NAME]
 [when CONDITION];
```

```
    PROCEDURES AND FUNCTIONS
PROCEDURES
    GET (A); PUT (A);
    NEW_LINE;
FUNCTIONS
    SORT (D);  -- FROM USER LIBRARY
```

NAMED vs POSITIONAL PARAMETERS

```
procedure CREATE
   (FILE: in out FILE_TYPE;
   MODE: in FILE_MODE := default;
   NAME: in STRING    := "  " ;
   FORM: in STRING    := "  ");
CREATE (WALDO);  --TEMP FILE
CREATE (JUDY, NAME => "JUDY");
```

OVERLOADING

```
PARAMETER_TYPE_PROFILE
  -- # PARAMETERS;
  -- BY POSITION, PARAMETERS
  -- HAVE SAME BASE TYPE
RESULT_TYPE_PROFILE
  -- SAME BASE TYPE
-- NOTE: NOT NAMES, NOT MODES,
-- NOT SUBTYPES, NOT DEFAULTS
```

OVERLOADING OPERATORS

- o CAN OVERLOAD PREDEFINED OPERATOR SYMBOLS
- o CANNOT OVERLOAD MEMBERSHIP TEST OR SHORT CIRCUIT CONTROL FORMS
- o EXAMPLE:

```
function "+" (LEFT,RIGHT: MATRIX)
  return MATRIX;
```

EXCEPTIONS

o DEAL WITH ERRORS OR OTHER EXCEPTIONAL SITUATIONS

o EXCEPTION NAMES ASSOCIATED WITH EXCEPTIONS AT COMPILE TIME AND STAY SAME NO MATTER HOW OFTEN DECLARATION IS ELABORATED

--EG, RECURSIVE CALLS DON'T

--PROLIFERATE EXCEPTIONS

RAISING EXCEPTIONS

- o DRAW ATTENTION TO ABNORMAL SITUATION
- o ABANDON NORMAL PROGRAM EXECUTION
- o TRANSFER CONTROL TO USER PROVIDED HANDLER
- o OR PROPAGATE

SUMMARY

FLOW OF CONTROL

1. SEQUENTIAL
2. IF. . .THEN. . .ELSE
3. CASE. . .IS. . .
4. WHILE. . .LOOP
5. FOR. . .LOOP
6. EXIT. . .WHEN
7. PROCEDURE CALL
8. FUNCTION INVOCATION
9. RETURN
10. RAISE --EXCEPTION

SUMMARY OF EXCEPTIONS

- o EXCEPTIONS ARE NOT OBJECTS, THEY ARE MERELY TAGS.
- o PROPAGATED DYNAMICALLY
- o THEY EXIST THROUGHOUT PROGRAM LIFE -
  - --CAN BE PROPAGATED OUT OF SCOPE AND THEN BACK IN AGAIN
- o ONLY USE FOR ABNORMAL EVENTS

EXAMPLE (CONT)

```
procedure AVERAGE is . . .
begin loop declare begin
  GET (DATA);
  if DATA = -1 then EXIT; end if;
  SUM := SUM + DATA
  COUNT := COUNT + 1;
  exception when CONSTRAINT_ERROR
   => BAD := BAD + 1; end;
  end loop; PUT ("AVERAGE =");
  PUT (FLOAT(SUM)/FLOAT(COUNT));
end AVERAGE;
```

EXAMPLE

```
procedure AVERAGE is
  DATA: INTEGER range -1..99_999;
  SUM, COUNT, BAD: INTEGER:=0;
begin
 loop                        --COLLECT DATA
   declare                   --TRAP BAD DATA
   begin ...                 --ACCUMULATE
   exception                 --COUNT BAD DATA
   end;                      --END ITERATION
 end loop;
end;
```

PROPAGATION OF EXCEPTIONS

WHERE RAISED AND NOT HANDLED	WHERE RAISED NEXT
SUBPROGRAM BODY	POINT OF CALL
BLOCK	AFTER BLOCK
PACKAGE BODY	AFTER BODY - WITHIN ENCLOSING DECLARATIVE PART
LIBRARY UNIT	ABANDON MAIN PROGRAM
TASK BODY	TASK COMPLETED

SCOPE OF EXCEPTIONS

- o EXCEPTIONS HAVE IDENTIFIERS
  A DEFINITION OF THE EXCEPTION IDENTIFIER MUST BE VISIBLE WHERE AN EXCEPTION IS RAISED AND WHERE A HANDLER IS DEFINED
- o HANDLERS ARE INDEPENDENT OF EXCEPTION DECLARATIONS AND ARE OPTIONAL

EXAMPLE

```
begin
  --SEQUENCE_OF_STATEMENTS
exception
  when SINGULAR|NUMERIC_ERROR=>
   PUT ("MATRIX IS SINGULAR");
  when others =>
   PUT ("FATAL ERROR");
   raise; --PROPAGATE SAME
            EXCEPTION
end;
```

EXCEPTION HANDLER

```
EXCEPTION HANDLER ::=
  when EXCEPTION_CHOICE
      {|EXCEPTION_CHOICE}
      =>SEQUENCE_OF_STATEMENTS
EXCEPTION_CHOICE ::=
  EXCEPTION_NAME | others
```

EXCEPTION HANDLERS

```
begin
  SEQUENCE_OF_STATEMENTS
exception
  EXCEPTION_HANDLER
   EXCEPTION_HANDLER
end
```

HANDLING EXCEPTIONS

```
declare
  N : INTEGER := 0;
begin
  N := N+J**A(K); --A&K GLOBAL
exception
  when others => PUT("AN ERROR");
end;
```

PREDEFINED EXCEPTIONS (CONT)

STORAGE_ERROR

- o DYNAMIC TASK STORAGE EXCEEDED
- o DURING ALLOCATION IF COLLECTION FULL
- o INSUFFICIENT STORAGE TO ELABORATE A DECLARATION OR CALL A SUBPROGRAM

PREDEFINED EXCEPTIONS (CONT)

PROGRAM_ERROR

- CALL SUBPROGRAM
- ACTIVATE TASK
- ELABORATE GENERIC

BEFORE BODY IS ELABORATED

- REACH END OF FUNCTION
- SELECTIVE WAIT WITHOUT OPEN BRANCHES
- ERRONEOUS ACTION
- INCORRECT ORDER DEPENDENCY

PREDEFINED EXCEPTIONS (CONT)

NUMERIC_ERROR

- EXECUTION OF PREDEFINED OPERATION CANNOT DELIVER CORRECT RESULT

TASKING_ERROR

- EXCEPTIONS DURING INTERTASK COMMUNICATIONS

PREDEFINED EXCEPTIONS

CONSTRAINT_ERROR

- VIOLATE RANGE CONSTRAINT
- VIOLATE INDEX CONSTRAINT
- VIOLATE DISCRIMINANT CONSTRAINT
- NON-EXISTENT RECORD COMPONENT
- NULL ACCESS VALUE

TAPE #5

DEFINING PROGRAM COMPONENTS

NEXT PAGE INTENTIONALLY
LEFT BLANK.

COMPONENT

- A VALUE THAT IS PART OF A LARGER VALUE
- AN OBJECT THAT IS PART OF A LARGER OBJECT

FOLLOWING ARE PROGRAM COMPONENTS

- o PROCEDURES
- o FUNCTIONS
- o PACKAGES
- o TASKS

SUBPROGRAM DECLARATION

```
procedure IDENTIFIER [formal_part]
function DESIGNATOR [formal_part]
   return type_mark
formal_part ::=
    (parameter_specification
    {; parameter_specification})
parameter_specification ::=
    identifier_list : mode
    type_mark [ := expression]
```

PROCEDURE EXAMPLES

```
procedure RIGHT_INDENT
   (MARGIN: out LINE_SIZE);
procedure SWITCH
   (FROM, TO: in out LINK);
procedure PRINT_HEADER
   (PAGES: in NATURAL);
    HEADER: in LINE
     := (1..LINE'LAST :='');
    CENTER:  in BOOLEAN := TRUE);
```

FUNCTION EXAMPLES

```
function RANDOM return
    PROBABILITY;
function MIN_CELL (X : LINK);
function DOT_PRODUCT
    (LEFT,RIGHT : VECTOR)
    return REAL;
function "*" (LEFT,RIGHT : MATRIX)
    return MATRIX;
```

PARAMETER PASSING MECHANISMS

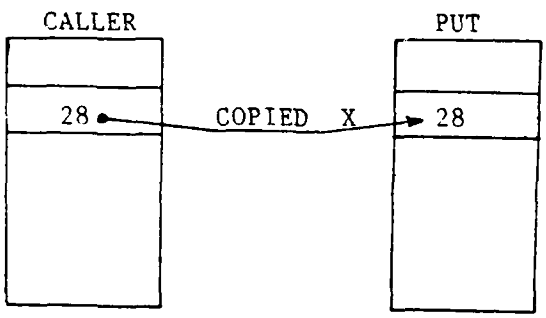
	in	in out	out
scaler	copy	copy	copy
access	copy	copy	copy
array	either reference or copy		
record			
task type			
private type	according to full type		

PARAMETER PASSING MECHANISMS

-COPY-

procedure PUT (X: INTEGER) is . . .

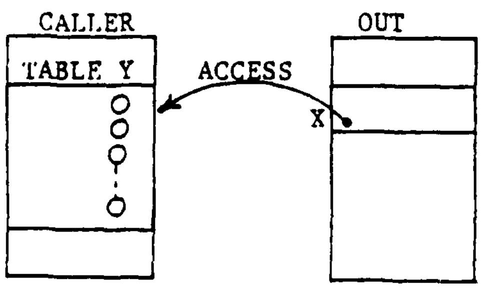
--CALLED BY PUT (28)



PARAMETER PASSING MECHANISMS

-REFERENCE-

```
type TABLE is array(1..10) of INTEGER;
  procedure OUT (X: TABLE) is . . .
  --CALLED BY
  -- Y : TABLE := (others=> 0);
  -- OUT (Y);
```



MATCHING ACTUALS & FORMALS

- o ACTUALS AND FORMALS MUST BE OF THE SAME BASE TYPF
- o IF FORMAL IS CONSTRAINED
  --ACTUAL VALUE MUST OBEY
  --CONSTRAINT
  --(BUT NOT TYPE OF ACTUAL)
- o IF ACTUAL <u>out</u> or <u>in out</u>
  --MORE CONSTRAINED THAN
  --FORMAL AND SCALER THEN
  --MUST OBEY AT COMPLETION

MATCHING ACTUALS & FORMALS

(CONT)

o IF FORMAL IS UNCONSTRAINED ARRAY

--ACTUAL MUST BE CONSTRAINED

--DETERMINES BOUNDS

o IF FORMAL IS RECORD OR PRIVATE

--WITH UNCONSTRAINED DISCRIMINANT

--THEN USE DISCRIMINANT OF

--ACTUAL INCLUDING UNCONSTRAINED

--IF ACTUAL IS UNCONSTRAINED

WARNINGS

- FOR ARRAYS AND RECORDS ASSIGNMENTS TO FORMAL MAY OR MAY NOT AFFECT ACTUAL IF SUBPROGRAM IS ABANDONED

- WHERE ACTUAL IS ACCESSIBLE BY MORE THAN ONE PATH (E.G. GLOBAL IDENTIFIER) VALUE IS UNDEFINED AFTER UPDATING BY ANY MECHANISM OTHER THAN ASSIGNING TO FORMAL AND RETURNING

```
SUBPROGRAM_DECLARATION ::=
    SUBPROGRAM_SPECIFICATION;
SUBPROGRAM_SPECIFICATION ::=
    procedure IDENTIFIER
       [FORMAL_PART]
   | function DESIGNATOR
       [FORMAL_PART]
       return TYPE_MARK
DESIGNATOR ::= IDENTIFIER
   | OPERATOR_SYMBOL
```

```
FORMAL_PART ::=
    (PARAMETER_SPECIFICATION
    {; PARAMETER_SPECIFICATION})
PARAMETER_SPECIFICATION ::=
    IDENTIFIER_LIST :
      MODE TYPE_MARK
      [ := EXPRESSION ]
MODE ::= [IN] | IN OUT | OUT
```

DISCRIMINANTS OF PRIVATE TYPES

```
type TEXT (MAX_LNG : INDEX)
     is limited private;
private
  type TEXT (MAX_LNG : INDEX) is
    record
      POS      : INDEX=o;
      VALUE    : STRING
        (1 .. MAX_LNG);
   end record;
end;
```

```
package KEY_MANAGER is
  type KEY is private;
  NULL_KEY : constant KEY;
  procedure GET_KEY
   (K : out KEY);
private
  type KEY is new NATURAL;
  NULL_KEY : constant := 0;
end;
```

PRIVATE TYPES (CONT)

- o PRIVATE PART OF INTERFACE
  - -- AFFECTS SPEARATE COMPILATION
- o LIMITED PRIVATE
  - -- NO IMPLICIT ASSIGNMENT
  - -- NO IMPLICIT EQUALITY
  - -- NO IMPLICIT INEQUALITY
- o DEFERRED CONSTANT
  - -- VALUE IS IN PRIVATE PART

PRIVATE TYPES (CONT)

<u>IMPLICITLY DEFINED OPERATIONS</u>

- o ASSIGNMENT
- o MEMBERSHIP TESTS
- o [DISCRIMINANT SELECTION]
- o EXPLICIT CONVERSIONS
- o T'BASE, T'SIZE, A'SIZE
  A'ADDRESS
- o [A'CONSTRAINED --
  IF DISCRIMINANT]
- o EQUALITY AND INEQUALITY

PRIVATE TYPES

- o PACKAGE DEFINES A SET OF OPERATIONS
- o PRIVATE TYPE DECLARATION CREATES A TYPE FOR OBJECTS TO WHICH THE OPERATIONS APPLY
- o DETAILS OF THE PRIVATE TYPE HIDDEN FROM USER

PRIVATE TYPES

- HIDE DETAILS OF TYPE DEFINITION
- ONLY OPERATIONS DEFINED BY PACKAGE AND THE IMPLICITLY DECLARED OPERATIONS CAN AFFECT PRIVATE TYPES

```
package WORK_DATA is
  type DAY is (MON, TUES, WED,
     THU, FRI, SAT, SUN);
  type HOURS is delta 0.25
     range 0.0 .. 24.0;
  type TIME_TABLE is
     array (DAY) of HOURS
  NORMAL : constant TIME_TABLE
     (MON..THR=> 8.25, FRI=> 7.0,
     OTHERS=> 0.0);
end WORK_DATA;
```

PACKAGE BODY

```
PACKAGE_BODY ::=
  package body SIMPLE_NAME is
    [DECLARATIVE_PART]
  [begin
    SEOUENCE_OF_STATEMENTS
 [exception
    EXCEPTION_HANDLER
    {EXCEPTION_HANDLER}]]
  end [SIMPLE_NAME];
```

PACKAGE SPECIFICATION

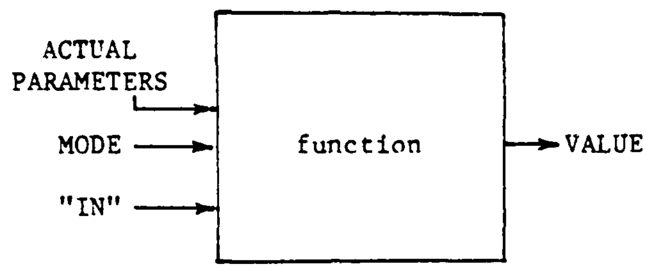
```
PACKAGE_SPECIFICATION ::=
  package IDENTIFIER is
    {BASIC_DECLARATIVE_ITEM}
  [  LIMTED] PRIVATE
    {BASIC_DECLARATIVE_ITEM}]
end [IDENTIFIER];
```

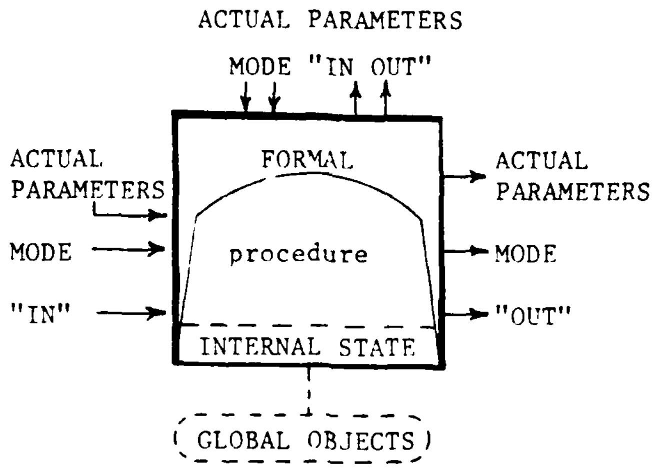
PACKAGE DECLARATION

```
PACKAGE_DECLARATION ::=
   PACKAGE_SPECIFICATION;
-- PACKAGE_BODY
-- IS OPTIONAL
```

PACKAGES

- A COLLECTION OF RELATED TYPES, SUBPROGRAMS, AND OBJECTS


ACTUAL
PARAMETERS
MODE
"IN"
function
VALUE


ACTUAL PARAMETERS
MODE "IN OUT"
ACTUAL PARAMETERS
MODE
"IN"
FORMAL
procedure
INTERNAL STATE
ACTUAL PARAMETERS
MODE
"OUT"
GLOBAL OBJECTS

PARAMETER MODES

o <u>IN</u>	FORMAL PARAMETER IS A CONSTANT
o <u>IN OUT</u>	CAN BOTH READ AND UPDATE ACTUAL
o <u>OUT</u>	CAN UPDATE ACTUAL CAN READ BOUNDS AND DISCRIMINANT OF ACTUAL (ONLY)

FOLLOWING ARE PROGRAM COMPONENTS

- o PROCEDURES
- o FUNCTIONS
- o PACKAGES
- o TASKS

LABORATORY #2

LABORATORY #2

OBJECTIVES:

1. EXPERIMENT WITH Ada CONSTRUCTS COVERED ON FIRST DAY
2. BUILD EXAMPLE FOR USE IN LABS 3 - 5

ASSIGNMENT

SIMULATE A SIMPLE
HAND CALCULATOR
IN Ada

CALCULATOR SPECIFICATIONS

```
type OPERATIONS is
  (+, -, *, /, C);
REGISTER : FLOAT;
--GET AN OPERATOR
--GET A NUMBER
--OPERATOR (REGISTER, NUMBER)
--PUT REGISTER
```

IMPROVE CALCULATOR

1. ROBUST HANDLING OF BLANK CHARACTERS
2. ACCEPT NUMBERS WITHOUT DECIMAL POINT
3. EXCEPTION HANDLER FOR INVALID INPUT

```
TYPE LEGAL_OPS
IS ('+', '-', '*', '/', c)
```

```
package OPS_IO
     is new
TEXT_IO. ENUMERATION_IO
  (LEGAL_OPS);
```

THIS PAGE BLANK

THIS PAGE BLANK

TYPE

o A SET OF VALUES

and

o A SET OF OPERATIONS
APPLICABLE TO
THOSE VALUES

TAPE #6

MAKING COMPONENTS MORE GENERAL

NEXT PAGE INTENTIONALLY
LEFT BLANK.

SUBTYPE

- SUBSET OF VALUES OF BASE TYPE
- DETERMINED BY A CONSTRAINT

OBJECT

- OBJECTS CONTAIN VALUES
- CREATED BY ELABORATING A DECLARATION (OR ...)
- TYPE BOUND AT ELABORATION

DISCRIMINANTS

```
type SQUARE (SIDE : INTEGER) is
  record
    MAT : MATRIX (1..SIDE,
                  1..SIDE);
  end record;
```

VARIANT RECORDS

AND DISCRIMINANTS

```
subtype DRUM_UNIT
  is PERIPHERAL (DRUM);
subtvpe DISK_UNIT
  is PERIPHERAL (DISK);
WRITER : PERIPHERAL
  (UNIT => PRINTER);
ARCHIVE : DISK_UNIT;
```

VARIANT RECORDS (CONT)

```
type PERIPHERAL
  (UNIT : DEVICE :=DISK) is
record STATUS : STATE;
  case UNIT is
     when PRINTER =>
       LINE COUNT : . . .
     when OTHERS =>
       CYLINDER : . . .
       TRACK : . . .
  end case;
end record;
```

VARIANT RECORDS

```
type DEVICE is
  (PRINTER, DISK, DRUM);
type STATE is
  (OPEN, CLOSED);
```

UNCONSTRAINED ARRAYS

```
function ROW_TOTAL (MATRIX)
 returns VECTOR is
begin
 if (MATRIX'LAST(2)-MATRIX'FIRST(2))
  /= (VECTOR'LAST-VECTOR'FIRST)
 then raise SOME_ERROR;
 --etc
```

REVIEW : UNCONSTRAINED ARRAYS

WRITE SUBPROGRAMS THAT DETERMINE SIZE OF FORMAL ARRAYS FROM ACTUAL

```
type MATRIX is array
  (INTEGER range <>,
   INTEGER range <>) of INTEGER;
type VECTOR is array
  (INTEGER range <>) of INTEGER;
```

QUESTIONS ON EXAMPLE

1. WHY NO NEW_LINE IN LOOP
2. WHAT IF DELETE exit
3. WHY NOT USE EXCEPTION AS TERMINAL CONDITION

TYPE ATTRIBUTES (CONT)

```
      NEW_PAGE;
      X :=P'FIRST;
PRINT : while X <= P'LAST loop
        for I in 1..NO_COL loop
          SET_COL ((I-1)*HT+1);
          PUT (P'IMAGE (X));
          exit PRINT when X=P'LAST;
          X:=P'SUCC(X);
        end loop; end loop PRINT;
```

TYPE ATTRIBUTES (CONT)

```
P'WIDTH MAXIMUM LENGTH OF
        IMAGES OF TYPE P
function LINE_LENGTH return COUNT;
NO_COL : constant INTEGER
     := LINE_LENGTH/(P'WIDTH+5);
HT := P'WIDTH+5;
```

TYPE ATTRIBUTES (CONT)

```
P'IMAGE (X);
  P IS DISCRETE SUBTYPE
  X IS A VALUE OF TYPE P
function P'IMAGE (X) returns STRING;
  RESULT IS THE PRINT IMAGE OF X
```

TYPE ATTRIBUTES

```
for J in BUFFER'RANGE loop
  if BUFFER (J) /= SPACE then
     PUT (BUFFER (J));
 end if;
end loop;
```

TOOLS FOR GENERALIZATION

- o TYPE ATTRIBUTES
- o VARIANT RECORDS
- o UNCONSTRAINED ARRAYS
- o UNCONSTRAINED DISCRIMINANTS
- o PACKAGES ARE ABSTRACT TYPES
- o GENERICS

OTHER STATEMENTS AFFECTING
FLOW OF CONTROL

10. RAISE --EXCEPTION
11. TASK INITIATION
12. ENTRY CALL -
    RENDEZVOUS FROM USER TASK
    LOOKS LIKE A PROCEDURE CALL
13. ACCEPT -
    RENDEZVOUS FROM SERVER TASK
14. ABORT
15. ~~GO TO~~ -- DON'T USE

FLOW OF CONTROL STATEMENTS

1. SEQUENTIAL
2. IF. . .THEN. . .ELSE
3. CASE. . .IS. . .
4. WHILE. . .LOOP
5. FOR. . .LOOP
6. EXIT. . .WHEN. . .
7. PROCEDURE CALL
8. FUNCTION INVOCATION
9. RETURN

DISCRIMINANTS

```
type TEXT (MAX_LNG : INDEX) is
  record
    POS : INDEX := 0;
    VALUE : STRING (1..MAX_LNG);
  end record;
```

UNCONSTRAINED DISCRIMINANTS

- DISCRIMINANTS MUST ALWAYS HAVE A VALUE
- NORMALLY, DISCRIMINANT IS A CONSTANT DETERMINED WHEN THE OBJECT IS CREATED
- HOWEVER, IF DEFAULT INITIAL VALUE GIVEN AND OBJECT CREATED BY NORMAL DECLARATION, DISCRIMINANT CAN BE CHANGED BY WHOLE RECORD ASSIGNMENT

STORAGE REQUIREMENT

KNOWN AT ALLOCATION

```
type TEXT (MAX_LNG:INDEX) . . .
LINE : TEXT(132);
type TEXT (MAX_LNG: INDEX:=80) . . .
CARD : TEXT;
LINE : TEXT(132); . . .
begin . . .
CARD :=LINE; --CHANGES MAX_LNG
```

PACKAGES AS ABSTRACT TYPES

```
package RATIONAL_NUMBERS is
  type RATIONAL is private
function EQUAL (X,Y:RATIONAL)
  return BOOLEAN;
function "+" (X,Y:RATIONAL)
  return RATIONAL;
function "-" (X,Y:RATIONAL)
  return RATIONAL;
--etc
```

PACKAGES AS ABSTRACT TYPES

```
package RATIONAL_NUMBERS is
 --etc (CONT.)
  procedure GET (ITEM:out RATIONAL;
                WIDTH:in FIELD:=0);
  procedure PUT (ITEM:RATIONAL;
    NUM_LNG:FIELD:=DEFAULT_NUM;
    DENOM_LNG:FIELD:=DEFAULT_DENOM)
```

GENERICS

```
generic
  type P is (<>); --DISCRETE
procedure LIST;
procedure LIST is
  --CODE TO LIST ALL
  --VALUES OF P
end LIST;
```

POWERFUL TOOLS FOR

THE LIBRARY BUILDER

- o ATTRIBUTES
- o VARIANT RECORDS
- o UNCONTRAINED FORMAL ARRAYS
- o UNCONSTRAINED RECORD DISCRIMINANTS
- o PACKAGES
- o GENERICS

NEXT PAGE INTENTIONALLY
LEFT BLANK.

TAPE #7

DESIGN GUIDELINES

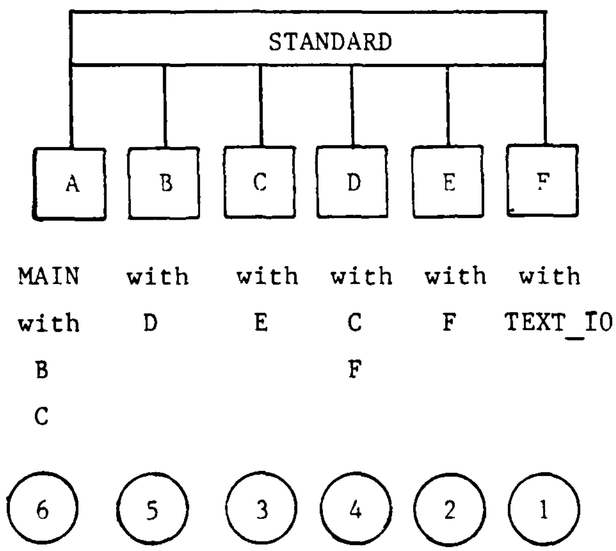
RULE #1

TRY TO STRUCTURE YOUR PROGRAM
AS A COLLECTION OF LIBRARY UNITS

RATIONALE:

1. LIBRARY UNITS ARE INDEPENDENT COMPONENTS
2. SIMPLIFIES SCOPE AND VISIBILITY

COMPILATION ORDER



RULE #2

PUT THE BODIES OF LIBRARY
UNITS IN SECONDARY UNITS

RATIONALE:

CHANGE IMPLEMENTATION OF A
LIBRARY UNIT WITHOUT
RECOMPILING UNITS USING IT

CREATING SECONDARY UNITS

LIBRARY UNITS:

- SUBPROGRAM DECLARATION/BODY
- PACKAGE DECLARATION
- GENERIC DECLARATION/INSTANTIATION

CREATING SECONDARY UNITS (CONT)

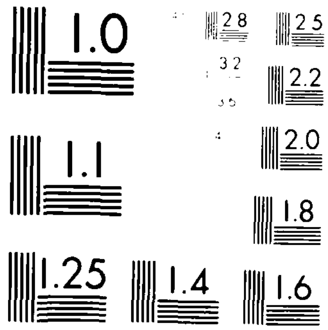
```
procedure EXAMPLE(X: in INTEGER,
                  Y: out STRING)
        is separate;
function SAMPLE(X: INTEGER)
        return STRING
        is separate;
```

CREATING SECONDARY UNITS (CONT)

```
package PROBLEM is                       SAME
      {DECLARATIONS}                     SOURCE
  [private                               FILE
      {OTHER_DECLARATIONS} ]              ⇓
  end                                    SAME
package body PROBLEM is                COMPILATION
  --etc                                    ?
```

AD-A155 779 AN INTRODUCTION TO ADA (TRADEMARK) FOR SCIENTISTS AND ENGINEERS(U) ARMY MATERIEL SYSTEMS ANALYSIS ACTIVITY ABERDEEN PROVING GROUND MD H E COHEN OCT 83 3/5

UNCLASSIFIED F/G 9/2 NL


1.0
2.8
2.5
3.2
2.2
2.0
1.1
1.8
1.25
1.4
1.6

RULE #3

CREATE A SYSTEM OF
MEANINGFUL TYPES
FOR YOUR APPLICATION

RATIONALE:

1. REDUCES CODING ERRORS
2. SIMPLIFIES MAINTENANCE
3. IMPROVES EXECUTION EFFICIENCY

CREATING MEANINGFUL TYPES:

DEFINING YOUR TYPES

```
package MY_TYPES is
  tvpe STOP_LIGHT is
    (READ, YELLOW, GREEN);
  tvpe DAY is
    (MON, TUE, WED, THU,
     FRI, SAT, SUN);
end MY_TYPES;
```

USING YOUR TYPES:

MAKING package MY_TYPES VISIBLE

```
with MY_TYPES;
procedure C is
  SIGNAL: MY_TYPES.STOPLIGHT;
begin
  --etc.
end C;
```

USING YOUR TYPES:

DIRECT VISIBILITY FOR INTERFACE OF

MY_TYPES

```
with MY_TYPES;
use MY_TYPES;
procedure C is
  SIGNAL : STOPLIGHT;
--etc.
```

USING YOUR TYPES:

OPERATIONS AVAILABLE ON USER DEFINED TYPES

1. EXPLICITLY DECLARED SUBPROGRAMS HAVING A PARAMETER OR RESULT OF THE TYPE
2. BASIC OPERATIONS
3. PREDEFINED OPERATORS
4. ENUMERATION LITERALS
5. ATTRIBUTES

missing in video

No change required

USING YOUR TYPES:

IMPLICITLY DECLARED OPERATIONS

<u>BASIC OPERATIONS INCLUDE:</u>

1. ASSIGNMENT AND INITIALIZATION
2. MEMBERSHIP TEXTS
3. QUALIFICATION - EG DAY'(MON)
4. EXPLICIT TYPE CONVERSION
5. IMPLICIT TYPE CONVERSION FOR UNIVERSAL_REAL AND UNIVERSAL_INTEGER

USING YOUR TYPES:

EXPLICITY DECLARED OPERATIONS

```
package RATIONAL_NUMBERS is
  tvpe RATIONAL is . . .
  function EOUAL (X,Y: RATIONAL)...
  function "/" (X,Y: INTEGER)...
  function "+" (X,Y: RATIONAL)...
--etc
```

DERIVED TYPES

ARE A CONVENIENT WAY TO

CREATE DISTINCT TYPES

EG:

```
type MEASURES is new RATIONAL;
```

DERIVES SUBPROGRAMS DECLARED

WITH THE PARENT TYPE

PITFALLS

RECOMPILING PACKAGE SPEC FORCES
RECOMPILATION OF ALL USING
PROGRAMS

```
--TO ADD MORE RATIONAL
  OPERATIONS
  with RATIONAL_NUMBERS;
  use RATIONAL_NUMBERS;
  package MORE_RATIONAL_OPS is
    function "*" . . .
```

OUTSIDE THE LANGUAGE

- o DIRECTORY SYSTEM FOR ORGANIZING
  - LIBRARY UNITS
  - (HIERARCHY OR RELATIONAL)
- o LIBRARY UNIT MANAGEMENT
  - SOURCE, OBJECT, EXECUTABLE MODULE
  - NEW AND DERIVED VERSIONS
  - WHICH PROGRAMS USE A UNIT

ASSUMPTIONS ABOUT COMPILER

- o UNUSED SUBPROGRAMS AND OBJECTS ELIMINATED
- o NO SPACE FOR ENUMERATION LITERALS UNLESS DISPLAYED
- o CONSTRAINTS CHECKED AT COMPILE TIME IF POSSIBLE

RULE #4

USE A PACKAGE TO ACHIEVE EFFECT OF FORTRAN COMMON (GLOBAL OBJECTS)

```
package COMMON is
  A,B,C: INTEGER;
end COMMON;
```

PACKAGE DECLARATIVE REGION

```
package WALDO is
  X,Y: ...              EXTEND TO SCOPE OF
end WALDO;              ENCLOSING DECLARATION
package body WALDO is
  X,Y: ...                         X,Y
begin                              EXIST
  SEQUENCE_OF_STATEMENTS           THROUGHOUT
end;                               WALDO
```

DECLARATIVE REGIONS

SUBPROGRAM	ENTRY & ACCEPT
PACKAGE	RECORD
TASK	RENAMING
GENERIC	BLOCK OR LOOP

OVERLOADING

<u>OVERLOADING</u>:

USING THE SAME IDENTIFIER FOR TWO OR MORE (HOPEFULLY RELATED) MEANINGS, WHERE THE MEANING IN A PARTICULAR SITUATION IS DETERMINED FROM THE CONTEXT

EXAMPLE:

```
"+"(LEFT,RIGHT : INTEGER)
   return INTEGER;
"+"(LEFT, RIGHT : FLOAT)
   return FLOAT;
```

SCOPE VS VISIBILITY

<u>SCOPE OF A DECLARATION:</u>

THE LIFETIME OF ANY ENTITY DECLARED BY THE DECLARATION

<u>VISIBILITY OF AN IDENTIFIER:</u>

IN COMBINATION WITH OVERLOADING, ESTABLISHES THE MEANING OF THE OCCURENCE OF AN IDENTIFIER

TAPE #8

SCOPE & VISIBILITY

ASSIGNMENT

1. CREATE A PACKAGE THAT IMPLEMENTS THE CALCULATOR FUNCTIONS
2. CREATE A MAIN PROCEDURE THAT HANDLES TERMINAL I-O AND USES THE CALCULATOR PACKAGE
3. SEPARATELY COMPILE BODIES OF ALL FUNCTIONS AND PROCEDURES

LABORATORY #3

OBJECTIVES:

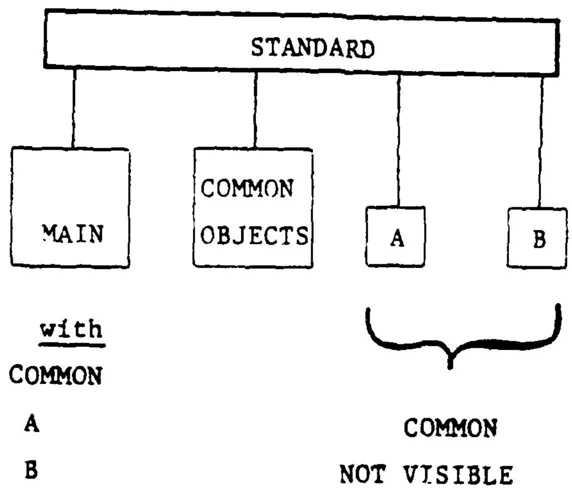
1. EXPERIMENT WITH PROCEDURE AND FUNCTION DEFINITIONS
2. DEFINE A PACKAGE
3. EXPERIMENT WITH SEPARATE COMPILATION

LABORATORY #3

SUMMARY

1. MAKE SUBSYSTEMS LIBRARY UNITS
2. SEPARATELY COMPILE LIBRARY BODIES
3. USE MEANINGFUL TYPES
4. PUT GLOBALS IN PACKAGE(S)
5. COMMUNICATE VIA PARAMETERS

RESTRICTED VISIBILITY



RULE #5

PASS INFORMATION THROUGH PARAMETERS RATHER THAN USING GLOBAL OBJECTS

COMMENTS:

1. DON'T ASSUME COMMUNICATING THROUGH GLOBAL OBJECTS IS MORE EFFICIENT
2. RESTRICT VISIBILITY OF GLOBAL DATA

USING NESTED PACKAGES (CONT)

```
with COMMON;
use COMMON;
procedure R is
use BLOCK_1;
--etc
```

USING NESTED PACKAGES

```
with COMMON;
procedure R is
use COMMON.BLOCK_1;
 --COULDN'T SAY IN CONTEXT
 --CLAUSE BECAUSE NOT
 --LIBRARY_UNIT_SIMPLE_NAME
```

NESTED PACKAGES

```
package COMMON is
   package BLOCK_1 is
      A,B,C : INTEGER;
   end BLOCK_1;
   package BLOCK_2 is
      X,Y,Z : FLOAT;
   end BLOCK_2
end COMMON;
```

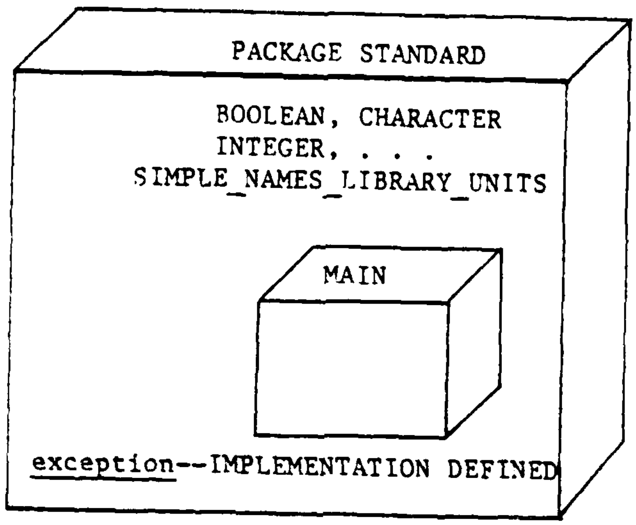
BLOCK DECLARATIVE REGION

```
WALDO : declare
        A :  . . .
        B :  . . .     SCOPE   SCOPE
  begin . . .           OF      OF
  exception . . .       A       B
  end WALDO;
```

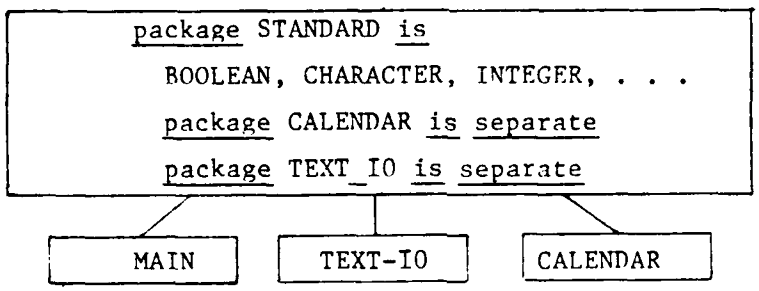
QUALIFIED NAMES

```
procedure P is
  A,B : BOOLEAN;
  procedure Q is
    C : BOOLEAN;
    B : BOOLEAN; --HOMOGRAPH OF P.B
  begin . . .
    B := A;  --Q.B := P.A;
    C := P.B; --Q.C := P.B;
```

NESTING HIERARCHY



NESTING HIERARCHY

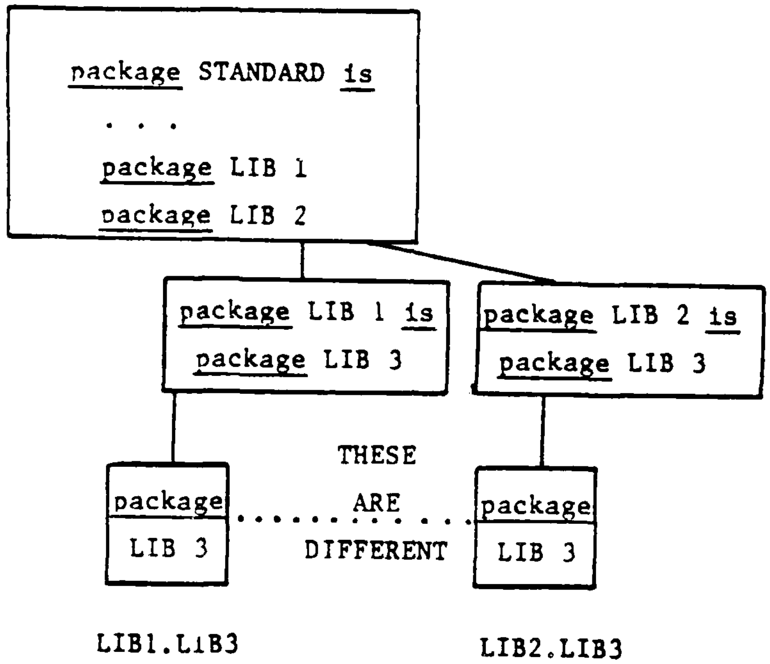


COMPONENT LIBRARY

o LIBRARY UNITS ARE IMPLICITLY
  DECLARED IN PACKAGE STANDARD
  --TO MAKE A LIBRARY UNIT VISIBLE:
    <u>with</u> LIBRARY_UNIT_SIMPLE_NAME;
  --NOTE IMPLICATION THAT <u>STANDARD</u>
    HAS UNUSUAL BEHAVIOR

COMPONENT LIBRARY (CONT)

- PROGRAM LIBRARY INCLUDES LIBRARY UNITS AND SECONDARY UNITS
- SECONDARY UNITS ARE SEPARATELY COMPILED SUB-UNITS OF LIBRARY UNITS OR OF OTHER SECONDARY UNITS
- LIBRARY UNIT SIMPLE NAMES MUST BE UNIQUE


package STANDARD is
. . .
package LIB 1
package LIB 2
package LIB 1 is
package LIB 3
package LIB 2 is
package LIB 3
package
LIB 3
THESE
ARE
DIFFERENT
package
LIB 3
LIB1.LIB3
LIB2.LIB3

USE CLAUSE

- ACHIEVES DIRECT VISIBILITY OF DECLARATIONS IN VISIBLE PARTS OF NAMED PACKAGES

```
package D is
   T,U,V : BOOLEAN;        ⎤  VISIBLE
   procedure P . . .       ⎬   PART
   procedure Q . . .       ⎦
end D;
```

USE CLAUSE (CONT)

- USE CLAUSE AFFECTS VISIBILITY BUT NOT SCOPE !
- SPECIAL CASES GUARANTEE THAT USE CANNOT HIDE AN OTHERWISE DIRECTLY VISIBLE DECLARATION:
  - --DIRECTLY VISIBLE HOMOGRAPHS HAVE PREFERENCE

USE AND RENAMES

o USE CLAUSE SHOULD BE VIEWED AS A CONVENIENT SHORTHAND

```
RENAMING_DECLARATION ::=
  IDENTIFIER : TYPE_MARK
    renames OBJECT_NAME;
 | IDENTIFIER : exception
    renames EXCP_NAME;
 | package IDENTIFIER
    renames PACK_NAME;
 | SUBPROG_SPEC
    renames SUB_PROG_OR_ENTRY;
```

RENAMING (CONT)

- RENAMING DOES <u>NOT</u> HIDE THE OLD NAME
- OBJECTS OF ANONYMOUS TYPE CANNOT BE RENAMED
- SUBTYPE CAN BE USED TO ACHIEVE EFFECT OF RENAMING TYPE:

```
subtype MODE is
  TEXT_IO.FILE_MODE;
```

OVERLOADING RULES

- DETERMINE ACTUAL MEANING OF IDENTIFIER WHEN VISIBILITY RULES SHOW MORE THAN ONE MEANING IS ACCEPTABLE
- ERR ON CONSERVATIVE SIDE - GIVE ERROR IF NOT ONE CLEARLY CORRECT INTERPRETATION

RESOLVING OVERLOADING CONFLICTS

- o USE EXPANDED NAMES WITH DOT NOTATION, EG

    MY_ARITH.SINE (X : FLOAT) . . .

- o USE QUALIFIED NAMES

    REAL'SINE (X : FLOAT) . . .

SUMMARY OF SCOPE AND VISIBILITY

- o DECLARATIONS ASSOCIATE IDENTIFIERS WITH ENTITIES
- o OFTEN, STORAGE ALLOCATION IS A SIDE EFFECT
- o IDENTIFIERS CAN HIDE OTHER IDENTIFIERS
- o GOAL IS TO MINIMIZE BOTH SCOPE AND VISIBILITY
  - --SCOPE RESTRICTIONS SAVE MEMORY
  - --VISIBILITY RESTRICTIONS MINIMIZE ERRORS

Ada RECORDS VS

FORTRAN, COBOL, DMS RECORDS

- o "RECORD" HISTORICALLY USED IN A PHYSICAL SENSE - A RECORD ON A STORAGE DEVICE
- o Ada RECORDS ARE AN ABSTRACTION - A LOGICAL GROUPING OF COMPONENTS

MODELING REALITY

```
type HOUSEHOLD
  (NUMBER_MEMBERS:POSITIVE := 1)
  is record
  HOME : DWELLING ;
  MEMBERS : array (1 ..
      NUMBER_MEMBERS)
    of PERSON;
  end record;
```

USE COMPOSITE TYPES

TO MODEL REALITY

```
type PERSON is record . . .
subtype STUDENT is PERSON;
type CLASS is array (POSITIVE
  range<>) of STUDENT;
type HOUSEHOLD
  (NUMBER_MEMBERS:POSITIVE :=1)
    is record . . .
```

FIXED LENGTH VS VARIABLE LENGTH
RECORDS

- o RECORD TYPES WITH NO DISCRIMINANT OR A CONSTRAINED DISCRIMINANT DEFINE A SET OF INDENTICAL OBJECTS
- o RECORD TYPES WITH UNCONSTRAINED DISCRIMINANT DEFINE LOGICALLY RELATED BUT (POTENTIALLY) DIVERSE SET OF OBJECTS
- o UNCONSTRAINED RECORDS CAN BE ARRAY COMPONENTS

FIXED LENGTH VS VARIABLE LENGTH
ARRAYS

- o NUMBER OF ARRAY COMPONENTS DETERMINED AT:
  - COMPILE TIME IF INDEX CONSTRAINT IS STATIC
  - TYPE DECLARATION ELABORATION IF INDEX CONSTRAINT IS DYNAMIC
  - OBJECT CREATION FOR UNCONSTRAINED ARRAYS
- o ALL ARRAY COMPONENTS ARE SAME SUBTYPE

COMPOSITE TYPES

A TYPE WHOSE VALUES
HAVE COMPONENTS

- ARRAYS
- RECORDS

TAPE #9

RECORD ABSTRACTION

NEXT PAGE INTENTIONALLY
LEFT BLANK.

THIS PAGE BLANK

```
package X is
 procedure Y is
begin
  declare




     declare
       procedire Z is
      declare
        procedure Z is
```

```
package COMMON IS
  I, J, K: INTEGER;
end COMMON;
with COMMON;
use COMMON;
procedure MINE is
    I: INTEGER;
 begin
   I:=I+1;
```

```
with TEXT_IO; TTY_IO;
use TEXT_IO; TTY_IO;
procedure P is
   package REAL_IO is
      new FLOAT_IO
         (FLOAT);
I: INTEGER;
begin
   GET (I);
```

```
procedure X is
   I : INTEGER
for I in 1..10 loop
  exit when.......
end loop
   PUT (I);
```

```
package TEST IS
  X : INTEGER;
procedure y is
  X:  INTEGER;
begin

end
end y;
```

THIS PAGE BLANK

Ada RECORDS VS

FORTRAN, COBAL, DMS RECORDS

TERMINOLOGY FROM THE PAST

- o FIXED LENGTH RECORD
- o VARIABLE LENGTH RECORD
- o RECORD TYPE
- o REPEATING GROUP
- o MASTER AND DETAIL RECORDS

Ada RECORDS VS

FORTRAN, COBOL, DMS RECORDS

1. EACH "OLD" RECORD IS A POSSIBLE IMPLEMENTATION OF AN _Ada_ ABSTRACT TYPE
2. EACH "OLD" RECORD CAN BE DESCRIBED IN _Ada_

EXAMPLES:

VARIABLE LENGTH RECORD

```
type VAR (LENGTH :
  POSITIVE := 1)
is record
V : STRING (1..LENGTH)
end record;
```

add

EXAMPLES:

MULTIPLE RECORD TYPES

```
subtype RECORD_ID is
  INTEGER range 1..99;
type RECORDS (KIND : RECORD_ID)
  is record
  case KIND is
    when 1 => . . .
    when 2 => . . .
  end case; end record
```

EXAMPLES

```
subtype RECORD_ID is
  (FAMILY, HOUSE, HEAD_OF_HOUSE,
  SPOUSE, CHILD, JOB, . . .)
type RECORDS (KIND:RECORD-ID)
  is record . . .
```

INPUT - OUTPUT

FOR FILES CONTAINING ELEMENTS
OF A GIVEN TYPE, USE GENERIC

SEQUENTIAL_IO

DIRECT_IO

SPECIFICATION OF THE

package DIRECT_IO

```
with IO_EXCEPTIONS;
generic
  type ELEMENT_TYPE is private;
package DIRECT_IO is
  type FILE_TYPE is limited private;
  type FILE_MODE is
    (IN_FILE, INOUT_FILE, OUT_FILE);
  type COUNT is range
    0..IMPLEMENTATION_DEFINED;
  subtype POSITIVE_COUNT is
    COUNT range 1..COUNT'LAST;
```

FILE MANAGEMENT

```
procedure CREATE
  (FILE : in out FILE_TYPE;
  MODE : in FILE_MODE := INOUT_FILE;
  NAME : in STRING := " ";
  FORM : in STRING := " ");
procedure OPEN
  (FILE : in out FILE_TYPE;
  MODE : in FILE_MODE;
  NAME : in STRING;
  FORM : in STRING := " ");
```

FILE MANAGEMENT (CONT)

```
procedure CLOSE
  (FILE : in out FILE_TYPE);
procedure DELETE
  (FILE : in out FILE_TYPE);
procedure RESET
  (FILE : in out FILE_TYPE;
    MODE : in FILE_MODE);
procedure RESET
  (FILE : in out FILE_TYPE)
```

FILE MANAGEMENT (CONT)

```
function MODE
  (FILE : in FILE_TYPE)
    return FILE_MODE;
function NAME
  (FILE : in FILE_TYPE)
    return STRING;
function FORM
  (FILE : in FILE_TYPE)
    return STRING;
function IS_OPEN
  (FILE : in FILE_TYPE)
    return BOOLEAN;
```

INPUT AND OUTPUT OPERATIONS

```
procedure READ (FILE : in FILE_TYPE;
  ITEM : out ELEMENT_TYPE;
     FROM : POSITIVE_COUNT);
procedure READ (FILE : in FILE_TYPE;
  ITEM : out ELEMENT_TYPE);
procedure WRITE (FILE : in FILE_TYPE;
  ITEM : in ELEMENT_TYPE;
    TO : POSITIVE_COUNT);
procedure WRITE (FILE : in FILE_TYPE;
  ITEM : in ELEMENT_TYPE);
```

INPUT AND OUTPUT (CONT)

```
procedure SET_INDEX
  (FILE : in FILE_TYPE;
     TO : in POSITIVE_COUNT);
function INDEX(FILE : in FILE_TYPE)
  return POSITIVE_COUNT;
function SIZE (FILE : in FILE_TYPE)
  return COUNT;
function END_OF_FILE
  (FILE : in FILE_TYPE)
    return BOOLEAN;
```

EXCEPTIONS

```
STATUS ERRORS: exception renames
  IO_EXCEPTIONS.STATUS_ERROR;
MODE_ERROR :exception renames
  IO_EXCEPTIONS.MODE_ERROR
NAME ERROR: exception renames
  IO_EXCEPTIONS.NAME_ERROR;
USE_ERROR :exception renames
  IO_EXCEPTIONS.USE_ERROR;
DEVICE_ERROR : exception renames
  IO_EXCEPTIONS.DEVICE_ERROR;
```

EXCEPTIONS (CONT)

```
END_ERROR   : exception renames
  IO_EXCEPTIONS.END_ERROR;
DATA_ERROR  : exception renames
  IO_EXCEPTIONS.DATA_ERROR;
private
  -- implementation-dependent
end DIRECT_IO;
```

ATTRIBUTES (CONT)

o P'SMALL = 2.0**(-P'EMAX-1) * $2.0^{-P'EMAX}$

$\overbrace{.1000...00}^{P'MANTISSA}$ -- = 0.5

= $2.0^{-1} * 2.0^{-P'EMAX}$

- 2.0**(-P'EMAX-1)

ATTRIBUTES (CONT)

o P'LARGE = 2.0**P'EMAX *

--MAX REQ EXP

(1.0-2.0**(-P'MANTISSA))

$$\underbrace{1.0-\overbrace{.000\ldots01}^{\text{P'MANTISSA}}}_{1.0-2.0^{(-\text{P'MANTISSA})}} = \overbrace{.111\ldots11}^{\text{P'MANTISSA}}$$

ATTRIBUTES (CONT)

o P'EPSILON = 2.0**(1-P'MANTISSA)

$$
\begin{aligned}
(1.0+\text{EPSILON}) &= \overbrace{.100 \;\ldots\; 1}^{\text{p'MANTISSA}} * 2^{1} \\
1.0 &= \underline{.100 \;\ldots\; 0 * 2^{1}} \\
\text{EPSILON} &= .000 \;\ldots\; 1 * 2^{1} \\
&= 2^{-\text{P'MANTISSA}} * 2^{1} \\
&= 2.0{**}(1-\text{P'MANTISSA})
\end{aligned}
$$

ATTRIBUTES

o P'DIGITS	# DECIMAL DIGITS IN MANTISSA
o P'MANTISSA	# BINARY DIGITS IN MANTISSA
o P'EMAX	4*P'MANTISSA - REQ EXPONENT RANGE
o P'EPSILON	MODEL NUMBERS ARE 1.0 & 1.0 + EPSILON
o P'LARGE	LARGEST NUMBER OF SUBTYPE P
o P'SMALL	SMALLEST NUMBER OF SUBTYPE P

DEFINING DECIMAL REALS

type REAL is digits 7;

--B = D*$\log_2 10$ - D*3.3219...

--B = 23.2533...=> 24 bit MANTISSA

--4*B = 93 => 7 bit EXPONENT

HOLE AROUND ZERO

$-.1001*2^{-16}$

$-.1000*2^{-16}$ $\quad \triangle = 2^{-20}$

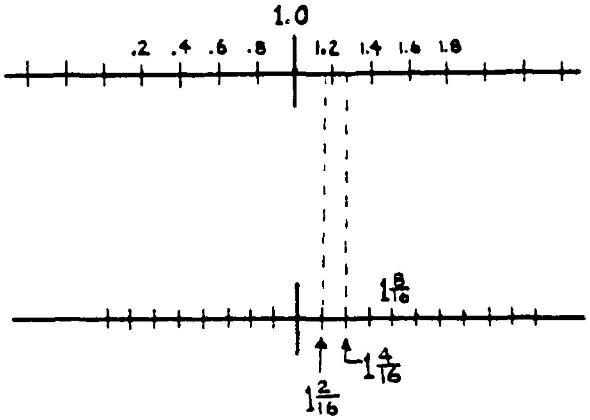
0

$.1000*2^{-16}$ $\quad \triangle = 2^{-17}$

$.1001*2^{-16}$

BE CAREFUL SUBTRACTING TWO
NEARLY EQUAL NUMBERS

FLOATING POINT



MODEL NUMBERS ARE

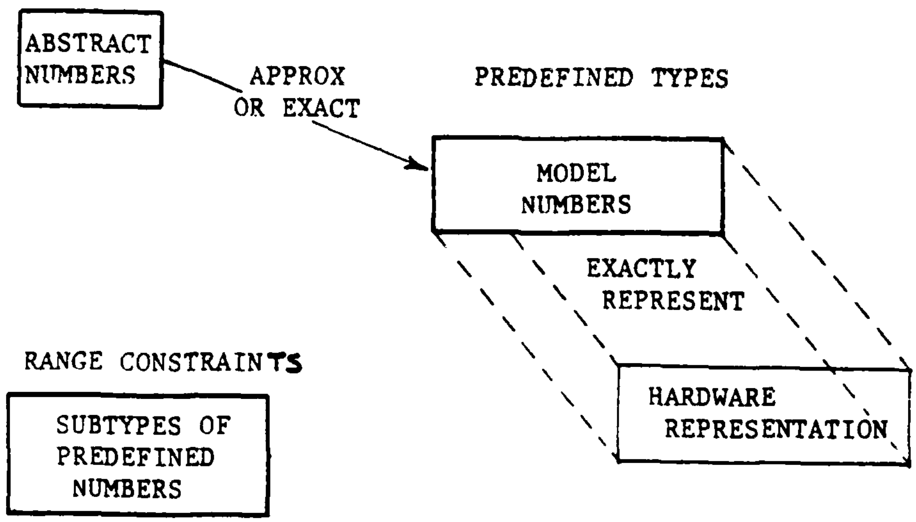
$$\left[\text{sign*MANTISSA*}2^{\text{EXPONENT}}\right]$$

$\frac{1}{2} \leq \text{MANTISSA} < 1$

INTEGERS

```
type MY_INTEGER
 is range-32_768..+ 32_767--16 BITS
MIGHT_MEAN
type X is new INTEGER; --32 BITS
type MY_INTEGER is X range
 -32_768..+32_767;
MODEL INTEGERS EXACTLY
REPRESENTED IN HARDWARE
```

THEORY AND IMPLEMENTATION



NUMERIC TYPES

```
UNIVERSAL_INTEGER
  INTEGER              --REQUIRED
  LONG_INTEGER         --OPTIONAL
  SHORT_INTEGER        --OPTIONAL
UNIVERSAL_REAL
  FLOAT
  FIXED
```

TAPE #10

NUMERIC ABSTRACTION

Next page intentionally
left blank.

SUMMARY

Ada RECORDS ARE VERY FLEXIBLE

USEFUL FOR MODELING REALITY

CAN BE USED FOR FILE I-O

Next page is intentionally left blank.

ATTRIBUTES

X' CONSTRAINED

TRUE IF X IS CONSTANT
OR HAS CONSTRAINED
DISCRIMINANT, FALSE
OTHERWISE

ALSO: BASE, FIRST BIT, LAST BIT,
POSITION, SIZE

NAMING AND COMPONENT SELECTION

X.Y.Z

TO SELECT COMPONENT Z

FROM RECORD Y, WHICH

IS A COMPONENT OF X

ATTRIBUTES (CONT)

o P'BASE'etc	--EMAX, LARGE, SMALL
	--FOR BASE TYPE
o P'MACHINE_	--EMAX, EMIN
	--MANTISSA
	--MACHINE_OVERFLOWS
	--MACHINE_RADIX
	--MACHINE_ROUNDS
o P'SAFE_	--EMAX, LARGE, SMALL

ATTRIBUTES (CONT)

```
o P'SAFE_      --EMAX, LARGE, SMALL
  P'BASE'EMAX   <=  P'SAFE_EMAX
  P'BASE'SMALL  >=  P'SAFE_SMALL
  P'BASE'LARGE  <=  P'SAFE_LARGE
```

DERIVED TYPES

- o DERIVED TYPE DEFINITION ::=
  new SUBTYPE INDICATION
- o SET OF VALUES FOR DERIVED TYPE
  IS COPY OF VALUES OF PARENT TYPE
- o CORRESPONDING BASIC OPERATONS,
  LITERALS, PREDEFINED OPERATORS,
  DERIVABLE SUBPROGRAMS
- o EXPLICIT TYPE CONVERSIONS

DERIVED TYPES VS SUBTYPES

- SUBTYPES RESTRICT THE
  SET OF VALUES;
  IMPLICIT CONVERSION AVAILABLE
- DERIVED TYPES ERECT A WALL
  SAME SET OF VALUES
  DIFFERENT ABSTRACT MEANING
  EXPLICIT CONVERSION REQUIRED

WHY DERIVED TYPES FOR NUMBERS

- o USER SPECIFIES ABSTRACT VALUES
- o COMPILER CHOOSES MOST
  APPROPRIATE MODEL TYPE
- o TWO TYPES DERIVED FROM SAME
  MODEL TYPE ON ONE MACHINE
  MAY USE DIFFERENT MODELS
  ON ANOTHER MACHINE
- o CONVERSIONS BETWEEN HARDWARE
  TYPES MAY BE EXPENSIVE
  SHOULD BE EXPLICITLY REQUESTED

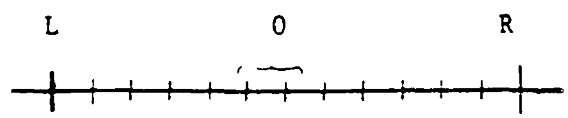
MANY OBJECTS OF SAME TYPE

```
type MEASUREMENTS
  is digits 3 --is FLOAT
  range 0.0 .. MEASUREMENTS'LARGE
A,B : MEASUREMENTS;
 C: MEASUREMENTS:
```

OBJECTS OF DIFFERENT TYPES

```
type LENGTH
 is digits 5  --is FLOAT
  range 0.0 .. 100.0
A : MEASUREMENTS;    BOTH DERIVED FROM
B:  LENGTH;          FLOAT ON THIS
                     MACHINE
```

FIXED POINT TYPES



tvpe T is delta D range L..R;

CANONICAL FORM ::=

  SIGN*MANTISSA*SMALL

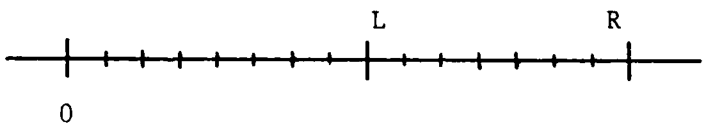
MANTISSA IS POSITIVE INTEGER

MODEL NUMBERS CHOOSE SMALL AS

  SMALL $= 2^{I} \leq D \leq 2^{I+1}$

--OR AS DEFINED IN LENGTH CLAUSE

MODEL FIXED POINT NUMBERS



- o MODEL NUMBERS INCLUDE ZERO
- o SMALL = $2^{I} \leq D \leq 2^{I+1}$
- o BOTH 'L AND 'R MUST BE WITHIN SMALL
  OF A MODEL NUMBER
- o CHOOSE SMALLEST NUMBER OF BITS IN MANTISSA
  -(B) SUCH THAT $-1 * 2^{B-1} *$ SMALL $\leq$ L

  $2^{B-1} *$ SMALL $\geq$ R

  AND (B) IS SMALLEST SUCH INTEGER

FIXED POINT VS FLOATING POINT

FIXED POINT W/DELTA=2	FLOATING POINT
1 $(*2^{-4})$	$.100.. *2^{-3}$
2 $(*2^{-4})$	$.100.. *2^{-2}$
3 $(*2^{-4})$	$.110.. *2^{-2}$
0 or 1	$.1 \quad *2^{-4}$
0	$.1 \quad *2^{-5}$
0	0.000

WORKING WITH VERY SMALL NUMBERS

FLOATING NUMBER NEAREST ZERO

= P'SAFE_SMALL

CHOOSE DELTA = $\lceil 2^{-(\text{P'SAFE\_EMAX} + 3)} \rceil$

THEN P'SAFE_SMALL = 50*DELTA

WORKING WITH VERY LARGE NUMBERS

LARGEST FLOATING NUMBER =

P'SAFE_LARGE

CHOOSE DELTA = $[2^{\text{P'SAFE\_EMAX} - 2}]$

THEN P'SAFE_LARGE = 2*DELTA

10 * P'SAFE_LARGE = 20*DELTA

PREDEFINED OPERATIONS

- o ADDITION AND SUBTRACTION

  OPERANDS MUST BE SAME TYPE

- o MULTIPLICATION AND DIVISION

  OPERANDS OF SAME TYPE

  ANY_FIXED_PT * INTEGER

  ANY_FIXED * ANY_FIXED =>

  UNIVERSAL_FIXED

- o EXPONENTIATION

  INTEGER TO INTEGER POWER

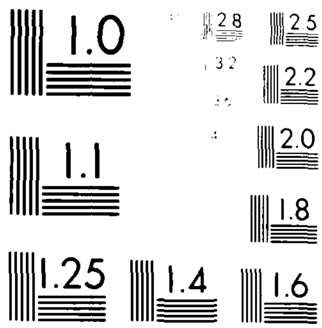
  FLOAT TO INTEGER POWER

CONVERSION

- o EXPLICIT TYPE CONVERSION DEFINED AMONG ALL NUMERIC TYPES
- o LITERALS ARE EITHER UNIVERSAL INTEGER OR UNIVERSAL REAL AND IMPLICITLY CONVERTED TO TYPE OF EXPRESSION WHERE USED

AD-A155 779 AN INTRODUCTION TO ADA (TRADEMARK) FOR SCIENTISTS AND ENGINEERS(U) ARMY MATERIEL SYSTEMS ANALYSIS ACTIVITY ABERDEEN PROVING GROUND MD H E COHEN OCT 83 4/5

UNCLASSIFIED F/G 9/2 NL


1.0
2.8
2.5
3.2
2.2
2.0
1.1
1.8
1.25
1.4
1.6

ATTRIBUTES OF FIXED POINT TYPES

T'DELTA	REQUESTED FIXED ACCURACY
T'SMALL	SMALLEST POSITIVE MODEL NUMBER
T'MANTISSA	NUMBER BINARY DIGITS IN MODEL NUMBERS
T'LARGE	LARGEST MODEL NUMBER
TFORE T'AFT	CHAR'S IN DECIMAL REPRESENTATION BEFORE AND AFTER DECIMAL POINT

ATTRIBUTES OF FIXED POINT TYPES

T'SAFE SMALL		SMALLEST POSITIVE NUMBER OF BASE TYPE OF T
T'SAFE LARGE		LARGEST NUMBER OF BASE TYPE
T'SAFE SMALL	=	T'BASE'SMALL
T'SAFE LARGE	=	T'BASE'LARGE
T'MACHINE ROUNDS		BOOLEAN
T'MACHINE OVERFLOWS		TRUE IF ALWAYS RECOGNIZES NUMERIC_ERROR

WHY POWER OF TWO FOR DELTA

```
type RULER is delta 2**(-4)
  range 0.0 .. 36.0;
type INCHES is new INTEGER
  range 0 .. 36;
FINE_MEASURE : RULER;
APPROX_MEASURE : INCHES;
FINE_MEASURE (APPROX_MEASURE)
  --LEFT SHIFT 4
APPROX_MEASURE (FINE_MEASURE)
  --ADD 8/16 AND RIGHT SHIFT 4
```

ALTERNATE VALUES FOR T'SMALL

```
type ENGLISH is delta 1.0/(12.0*16.0)
  range 0.0 .. MAX_INT/1024;
for ENGLISH'SMALL use 1.0/(12.0*16.0)
  delta 1.0/12.0;
type FEET is new INTEGER
--EVERY SIXTEENTH MODEL NUMBER
--OF ENGLISH IS A MODEL NUMBER
--OF INCHES
```

ALTERNATE VALUES FOR T'SMALL

```
type MONEY is delta 1.0/1000.0
  range 0.0 .. MAX_INT/1024;
for MONEY'SMALL use 1.0/1000.0;
subtype PENNIES is MONEY
  delta 1.0/100.0;
subtype DIMES is MONEY
  delta 1.0/10.0;
subtype QUARTERS is MONEY
  delta 1.0/4.0;
```

SUMMARY OF FIXED POINT

- FIXED POINT PROVIDES FOR RATIONAL ARITHMETIC ON MODEL NUMBERS
- COMMON DENOMINATOR IS 1/T'BASE'SMALL
- ADDITION AND SUBTRACTION ARE EXACT AS LONG AS ACTUAL VALUES ARE MODEL NUMBERS

SUMMARY OF FIXED POINTS (CONT)

- DEFAULT T'BASE'SMALL IS POWER OF TWO, WHICH IS EQUIVALENT TO FLOATING POINT WITH CONSTANT EXPONENT
- MULTIPLICATION AND DIVISION PRODUCE UNIVERSAL_REAL, WHICH MUST BE CONVERTED
- CONVERSION CAN INTRODUCE ROUNDOFF ERRORS UNLESS SOURCE AND TARGET DERIVED FROM SAME BASE TYPE

SUMMARY

- NUMERIC TYPES ARE:
  - INTEGER
  - FLOAT } REAL
  - FIXED
- USER DEFINED TYPES ARE DERIVED FROM BUILT-IN (HARDWARE) NUMERIC TYPES

LABORATORY #4

LABORATORY #4

OBJECTIVES:

1. IMPLEMENT PARENTHETICAL EXPRESSIONS IN YOUR HAND CALCULATOR USING A RECORD WITH DISCRIMINANT TO SAVE INTERMEDIATE RESULTS
2. ADJUST "IS DIGITS" UNTIL YOU CAN DEMONSTRATE ROUND-OFF ERROR

ASSIGNMENT

1. IMPLEMENT PARENTHETICAL EXPRESSIONS IN YOUR HAND CALCULATOR USING A RECORD WITH DISCRIMINANT TO SAVE INTERMEDIATE RESULTS
2. ADJUST "IS DIGITS" UNTIL YOU CAN DEMONSTRATE ROUND-OFF ERROR

Next page intentionally left blank.

TAPE #11

REVIEW

NEXT PAGE INTENTIONALLY
LEFT BLANK.

STACK (CONT)

```
package body STACK is
  type TABLE is array
    (POSITIVE range<>) of ITEM;
  SPACE : TABLE(1..SIZE);
  INDEX : NATURAL := 0;
  procedure PUSH(E: in ITEM) is
  begin
```

GENERIC STACK

```
generic
   SIZE : POSITIVE;
   type ITEM is private;
package STACK is
   procedure PUSH (E: in ITEM);
   procedure POP (E: out ITEM);
   OVERFLOW, UNDERFLOW : exception
end STACK;
```

PACKAGE ON_VECTORS (CONT)

```
function SIGMA(A: VECTOR)
  return ITEM is
  TOTAL : ITEM := A(A'FIRST);
    --THE FORMAL tvpe ITEM
begin
  for N in A'FIRST+1..A'LAST loop
    TOTAL :- SUM(TOTAL,A(N));
      --THE FORMAL function SUM
  end loop;
  return TOTAL;
end;
```

PACKAGE ON_VECTORS (CONT)

```
begin
  if A'LENGTH /= B'LENGTH then
    raise LENGTH_ERROR;
  end if;
  for N in A'RANGE loop
    RESULT(N):=SUM(A(N),B(N+BIAS));
     --THE FORMAL function SUM
  end loop;
  return RESULT;
end;
```

PACKAGE ON_VECTORS (CONT)

```
function SUM (A,B: VECTOR)
  return VECTOR is
  RESULT : VECTOR (A'RANGE);
    --THE FORMAL type VECTOR
  BIAS : constant
    INTEGER :=B'FIRST - A'FIRST;
```

GENERIC INSTANTIATION

```
package INT_VECTOR is new
  ON_VECTORS
    (INTEGER,
     TABLE,
     "+");
```

ON_VECTORS (CONT)

```
package ON_VECTORS is
  function SUM (A,B: VECTOR)
   return VECTOR;
  function SIGMA (A:VECTOR)
   return ITEM;
  LENGTH_ERROR: exception;
end;
```

GENERIC PACKAGE EXAMPLE

```
generic
  type ITEM is private;
  type VECTOR is array
    (POSITIVE range <>)
    of ITEM;
  with function SUM (X,Y: ITEM)
    return ITEM;
package ON_VECTOR is
```

GENERIC EXAMPLE

```
generic
  type ELEM is private;
procedure EXCHANGE
  (U,V: in out ELEM);
procedure EXCHANGE
  (U,V: in out ELEM) is
  T: ELEM;
begin T:=U; U:=V; V:=T;
end;
```

GENERICS

DEFINE A TEMPLATE FOR EITHER

SUBPROGRAMS OR PACKAGES

GENERIC FORMAL PARAMETERS CAN BE

- OBJECTS
- TYPES
- SUBPROGRAMS

TAPE #12

GENERICS

KEY TERMS (CONT)

- o ABSTRACT VS PHYSICAL REPRESENTATION
- o NUMERIC EXCEPTIONS
- o ATTRIBUTES OF NUMBERS
- o BINDING - BINDING TIME
- o SOFTWARE COMPONENT

KEY TERMS (CONT)

- o DISCRETE RANGE
- o CONSTRAINED VS UNCONSTRAINED
- o WHOLE RECORD ASSIGNMENT
- o UNIVERSAL REAL
- o UNIVERSAL INTEGER

KEY TERMS

- o SYNTAX DIAGRAMS
- o DERIVED TYPES VS SUBTYPES
- o OVERLOADING VS HIDING
- o USE VS WITH
- o DISCRIMINANT
- o TYPE MARK

STACK (CONT)

```
procedure PUSH(E: in ITEM) is
begin
  if INDEX >= SIZE then
    raise OVERFLOW;
  end if;
  INDEX := INDEX + 1;
  SPACE(INDEX) := E;
end PUSH;
```

STACK (CONT)

```
procedure POP(E: out ITEM) is
begin
  if INDEX = 0 then
    raise UNDERFLOW;
  end if;
  E := SPACE(INDEX);
  INDEX := INDEX - 1;
end POP;
end STACK
```

STACK (CONT)

```
package STACK_INT is new
 STACK(SIZE =>200,
 ITEM =>INTEGER);
package STACK_BOOL is new
 STACK(100, BOOLEAN);
STACK_INT.PUSH(N);
STACK_BOOL.PUSH(TRUE);
```

STACK (CONT)

```
generic
  type ITEM is private
package ON_STACKS is
  type STACK(SIZE : POSITIVE)
    is limited private;
  procedure PUSH(S : in out STACK;
    E : in ITEM);
  procedure POP(S : in out STACK;
    E : out ITEM);
  OVERFLOW, UNDERFLOW : exception ;
```

```
private
  type TABLE is array
    (POSITIVE range<>) of ITEM;
  type STACK(SIZE : POSITIVE) is
    record
      SPACE : TABLE(1..SIZE);
      INDEX : NATURAL := 0;
    end record;
end;
```

STACK (CONT)

```
declare
  package  STACK_REAL is new
    ON_STACKS(REAL);
      use STACK_REAL;
  S : STACK(100);
begin
  . . .
  PUSH(S, 2.54);
  . . .
end;
```

GENERIC INSTANTIATION
INSTANTIATED IN SCOPE OF
DECLARATION, NOT WHERE
IT IS INSTANTIATED

SUMMARY OF GENERICS

- o GENERICS ARE USED TO CREATE TEMPLATES FOR COMMON FUNCTIONS
- o GENERICS WILL BE WIDELY USED IN Ada
- o ALREADY, GENERICS USED FOR:

  INTEGER_IO

  FLOAT_IO

  FIXED_IO

  ENUMERATION_IO

NEXT PAGE INTENTIONALLY
LEFT BLANK.

TAPE #13

ACCESS TYPES

NEXT PAGE INTENTIONALLY
LEFT BLANK.

ACCESS TYPES

USED WHEN BLOCK STRUCTURED

ALLOCATION OF OBJECTS AND

OBJECT NAMES IS TOO RESTRICTIVE

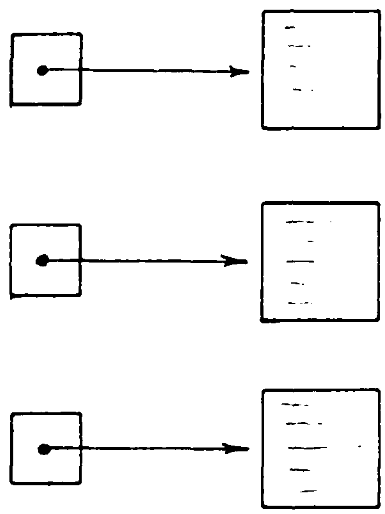
REVIEW OF BLOCK STRUCTURED OBJECTS:

- o NAMES BOUND TO OBJECTS
- o OBJECTS DESTROYED WHEN CONTROL LEAVES DECLARING UNIT
- o NUMBER OF OBJECTS DETERMINED WHEN DECLARATIONS ARE ELABORATED

ACCESS TYPES

OBJECTS OF ACCESS TYPES

ACCESSED OBJECTS



SCOPE OF ACCESSED OBJECTS

SCOPE IS THAT OF
THE ACCESS TYPE

ACCESSED OBJECTS
FORM A "COLLECTION"

OBJECTS BECOME INACCESSIBLE
WHEN NO VARIABLES REFER
TO THEM DIRECTLY
OR INDIRECTLY

REFERRING TO ACCESSED OBJECTS

o P.NEXT --POINTER FIELD OF

--RECORD ACCESSED BY P

o P.all --COMPLETE RECORD

--ACCESSED BY P

o Q:POINTER :=P;

--P & Q POINT TO SAME RECORD

```
procedure SHOPPING is
  task MOTHER;
  task body MOTHER is
    begin BUY_MEAT; end MOTHER;
  task BOY;
  task body BOY is
    begin BUY_BEER; end BOY;
begin BUY_GAS; end SHOPPING;
```

```
procedure SHOPPING is
  task GET_MEAT; --BY MOTHER
  task body GET_MEAT is
    begin BUY_MEAT; end GET_MEAT;
  task GET_BEER; --BY SON
  task body GET_BEER is
    begin BUY_BEER: end GET_BEER;
begin BUY_GAS; end SHOPPING;
```

```
task T is --SPECIFICATION
  . . .
end T;
task body T is --BODY
  . . .
end T;
```

TASKS

- TASKS ARE USED TO EXPRESS INDEPENDENT ACTIVITIES
- TASKS CAN (BUT MAY NOT) BE EXECUTED IN PARALLEL WITH EACH OTHER
- THE RENDEZVOUS IS USED TO SYNCHRONIZE TASKS AND TO EXCHANGE DATA BETWEEN TASKS

TAPE #14

TASK/TASK TYPES

NEXT PAGE INTENTIONALLY
LEFT BLANK.

SUMMARY OF ACCESS TYPES

- o ACCESS TYPES PROVIDE VERY FLEXIBLE CAPABILITY TO CREATE AND DESTROY OBJECTS
- o STRONG TYPING IS ENFORCED, SINCE EACH ACCESS TYPE IS TIED TO A SINGLE COLLECTION OF ACCESSED OBJECTS
- o CANNOT BE DANGLING POINTERS SINCE INITIALIZED TO <u>null</u> AND ASSIGNMENTS TYPE CHECK (UNLESS ERRONEOUS UNCHECKED_DEALLOCATION)

MAILBOX EXAMPLE (CONT)

```
or accept RECEIVE (RESPONSE:out
     STRING(<>)) do
  if LAST-null then
       RESPONSE:="b";
  else RESPONSE := LAST.TEXT;
  X := LAST; LAST := LAST.PRIOR;
  FREE(X); end if;
end select; end loop; end MAILBOX;
```

MAILBOX EXAMPLE (CONT)

```
begin loop select
  accept SEND(MESSAGE:STRING
     (<>)) do
    X := FIRST;
    FIRST := new NODE'
      (MESSAGE, null);
    if LAST = null
      then LAST := FIRST;
      else X.PRIOR := FIRST
   end if;
```

MAILBOX EXAMPLE (CONT)

```
X, FIRST, LAST: POINTER :
   --INITIALLY null
  procedure FREE is new
    UNCHECKED_DEALLOCATION
    (NODE,POINTER);
```

MAILBOX EXAMPLE (CON'T)

```
type NODE (SIZE:INTEGER
  range 1..132) is
  record
    TEXT : STRING (1..SIZE);
    PRIOR : POINTER;
  end record;
```

UNBOUNDED MAILBOX EXAMPLE

```
task MAILBOX is
  entry SEND (MESSAGE: STRING (<>));
  entry RECEIVE (RESPONSE:out
    STRING (<>));
end MAILBOX;
task body MAILBOX is
  type NODE;
  type POINTER is access NODE;
  --etc.
```

```
QUALIFIED_EXPRESSION ::=
  TYPE_MARK' (EXPRESSION)
 |TYPE_MARK'AGGREGATE
```

INITIALIZING A NODE

```
P1 : POINTER:= new NODE'
     (21, null);
-- ALLOCATOR new
-- TAKES QUALIFIED EXPRESSION
-- RETURNS ACCESS VALUE
-- THAT DESIGNATES OBJECT
```

LIST PROCESSING EXAMPLE

```
type NODE;
type POINTER is access NODE;
type NODE is
  record
     VALUE : INTEGER;
     NEXT : POINTER;
  end record
P : POINTER; --DEFAULT VALUE null
```

TASKS : JOBS TO BE DONE

GET_MEAT

GET_BEER

GET_GAS

PROCESSORS : RESOURCES FOR PERFORMING TASKS

MOTHER

CHILDREN

DECLARATIONS ARE

ELABORATED

DURING PROGRAM EXECUTION

- o INTRODUCE IDENTIFIERS
- o ALLOCATE STORAGE
- o INITIALIZE OBJECTS

OBJECTS DESTROYED AT END

OF BLOCK WHERE DECLARED

DECLARATION,
ACTIVATION,
TERMINATION

- TASKS ARE COMPONENTS, DECLARED IN SUBPROGRAM, BLOCK, PACKAGE, TASK_BODY
- ACTIVATE WHEN PARENT HITS <u>begin</u>
- TERMINATE AT FINAL <u>end</u>

RENDEZVOUS

o ONE TASK CALLS ENTRY IN ANOTHER

```
     task T is
       entry E (...);
     end;
   --etc
     T.E (...);
```

IMPLEMENTING ENTRIES

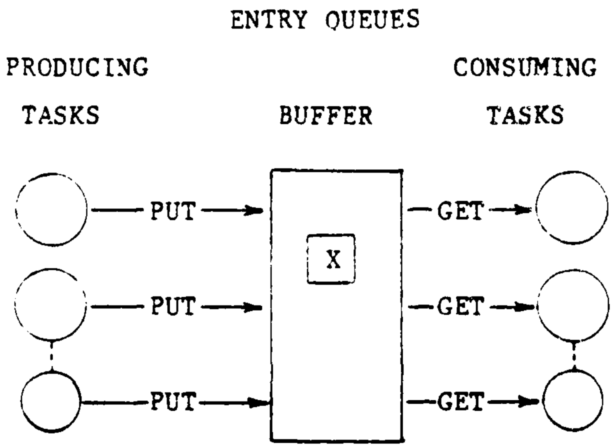
```
task T is
  entry E (...);
end;
task body T is
  entry E (...);
begin;
  accept E (...) do
  --SEQUENCE_OF_STATEMENTS
  end E;
end T;
```

ENTRY QUEUES

```
task BUFFER is
  entry PUT(X:in ITEM);
  entry GET(X:out ITEM);
end;
```

```
task BUFFER is
  entrv PUT (X:in ITEM);
  entrv GET (X:out ITEM);
end;

  task bodv BUFFER is
    V:ITEM;
  begin loop
    accept PUT (X:in ITEM) do
      V:=X; end PUT;
    accept GET (X:out ITEM) do
      X:V; end GET;
  end loop;
  end BUFFER;
```


ENTRY QUEUES
PRODUCING TASKS
BUFFER
CONSUMING TASKS
PUT
PUT
PUT
X
GET
GET
GET

ENTRY QUEUES

- o FIRST IN - FIRST OUT (FIFO)
- o WITHIN TASK BODY, ATTRIBUTE E'COUNT TELLS NUMBER OF WAITING TASKS

ASYMMETRIES

- o CALLER MUST NAME CALLED TASK
  ENTRIES ARE AVAILABLE TO ANY TASK
- o ENTRY CAN HAVE SEVERAL QUEUED
  TASKS
  TASK CAN ONLY BE ON ONE QUEUE
- o TASK NAMES CANNOT APPEAR IN
  USE CLAUSE, SO NEED DOT NOTATION
- o SOURCE TEXT FOR A TASK CAN
  CONTAIN MORE THAN ONE ACCEPT
  FOR AN ENTRY; THEY ALL DRAW
  FROM SAME QUEUE

TIMING AND SCHEDULING

IF TWO TASKS WITH DIFFERENT PRIORITIES ARE ELIGIBLE FOR EXECUTION AND COULD SENSIBLY USE SAME RESOURCES, LOWER PRIORITY TASK CANNOT EXECUTE WHILE HIGHER PRIORITY TASK WAITS

```
pragma PRIORITY (integer);
```

SCHEDULING

- o IMPLEMENTATION SHOULD MAKE AN "ARBITRARY" CHOICE AMONG EQUAL PRIORITY TASKS
- o IMPLEMENTATION CAN CHOOSE AN EFFICIENT (BUT FAIR) MECHANISM
- o PROGRAMS THAT DEPEND ON SELECTION ALGORITHM ARE ERRONEOUS

PRIORITY

- THE PRIORITY IN A PRIORITY *pragma* MUST BE A STATIC EXPRESSION THAT EVALUATES TO AN INTEGER
- LARGER INTEGERS IMPLY GREATER URGENCY
- THE RANGE OF SUBTYPE PRIORITY IS IMPLEMENTATION DEFINED

PRIORITY

- PRIORITY IS OPTIONAL
- PRIORITY OF RENDEZVOUS IS GREATER OF TWO PRIORITIES
- IF NO EXPLICIT PRIORITY, SCHEDULING RULES NOT DEFINED
- TASK PRIORITIES
  STATIC = CONSTANT

TIMED ENTRY CALL EXAMPLE

```
select
  CONTROLLER.REQUEST
     (MEDIUM) (SOME_ITEM);
or
  delay 45.0;
  -- give up
end select;
```

CONDITIONAL ENTRY CALL EXAMPLE

```
-- BUSY_WAIT
procedure SPIN (R: RESOURCE) is
begin loop select
            R.SEIZE;
            return;
        else;
            null; --busy wait
        end select;
    end loop;
end;
```

USER SELECTS

- o CONDITIONAL ENTRY CALL

  CANCEL CALL IF RENDEZVOUS

  NOT IMMEDIATELY POSSIBLE

- o TIMED ENTRY CALL

  CANCEL CALL IF NO RENDEZVOUS

  BEFORE TIMER RUNS OUT

ONLY ON ONE ENTRY AT A TIME!!

MECHANICS OF SELECT

- GUARDS (WHEN CONDITION =>) EVALUATED WHEN SELECT IS EXECUTED
- ABSENT GUARD IS TRUE
- ORDER OF EVALUATION OF GUARDS IS UNDEFINED
- SELECT ERROR IF ALL GUARDS ARE FALSE

```
select
  when COUNT < N   =>
    accept PUT . . .
  or
  when COUNT > 0   =>
    accept GET (X : out ITEM) do

       x := A(J);
     end;
     J := J mod N+1;
     COUNT := COUNT - 1;

end select;
```

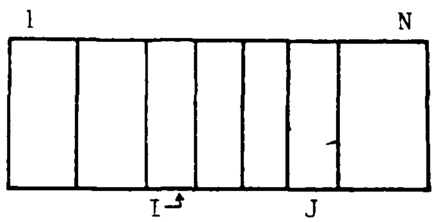
```
task body BUFFER is
  N : constant := 8; --EXAMPLE
  A : array (1..N) of ITEM;
  I,J : INTEGER range 1..N := 1;
  COUNT : INTEGER range 0..N :=0;
begin loop

select
  when COUNT < N  =>
  accept PUT (X : in ITEM) do
    A(I) := X;
  end;
  I := I mod N+1:
  COUNT := COUNT+1;
or
```

```
task body BUFFER is
  N : constant := 8; --EXAMPLE
  A : array (1..N) of ITEM;
  I,J : INTEGER range 1..N := 1;
  COUNT : INTEGER range 0..N :=0;
begin loop
```

BUFFER EXAMPLE

```
task BUFFER is
  entry PUT (X : in ITEM);
  entry GET (X : out ITEM);
end;
```



SERVER SELECTS

```
select
  SELECT_ALTERNATIVE
{or
  SELECT_ALTERNATIVE
[else
  SEQUENCE_OF_STATEMENTS]
end select;
```

EXAMPLE (SELECT STATEMENT)

```
select
  accept DRIVER_AWAKE_SIGNAL;
or
  delay 30.0 * SECONDS;
  STOP_THE_TRAIN;
end select;
--SELECTIVE WAIT CAN HAVE MORE
--THAN ONE DELAY ALTERNATIVE
```

SELECT STATEMENT

- o SELECT FIRST OF SEVERAL ENTRIES TO BE CALLED
- o MAKE AVAILABILITY OF AN ENTRY CONDITIONAL
- o CONTROL WAITING TIMES

SYNCHRONOUS EXECUTION

```
declare use CALENDAR;
  INTERVAL:constant DURATION:=1.0;
  NEXT_TIME: TIME:=FIRST_TIME;
begin loop
  delay NEXT_TIME - CLOCK ();
  ACTION;
  NEXT_TIME:=NEXT_TIME+INTERVAL;
end loop; end;
```

DELAY STATEMENT

```
delay DURATION;
type DURATION
--IMPLEMENTATION DEFINED
--FIXED POINT TYPE
--IN SECONDS
--AT LEAST ONE DAY (86_400)
delay 3.0; --AT LEAST 3 SECONDS
```

SYNCHRONIZATION

- USE RENDEZVOUS FOR SYNCHRONIZATION
- USE PRIORITIES TO INDICATE RELATIVE URGENCY AND FINE TUNE RESPONSIVENESS

TIMED ENTRY CALL

```
TIMED_ENTRY_CALL::=
  select
    ENTRY_CALL_STATEMENT
    [SEQUENCE_OF_STATEMENTS]
  or
    DELAY_ALTERNATIVE
  end select;
```

SUMMARY OF TASKING

TASKS VS PROCESSORS

TASK DECLARATIONS

TASKS INITIATION

(DECLARATION ELABORATION)

TASK TERMINATION

RENDEZVOUS_SYNCHRONIZATION

TASK TYPES

TASK TYPES

- ARE USED TO CREATE SEVERAL SIMILAR BUT DIFFERENT TASKS
- TASK OBJECTS BEHAVE AS CONSTANTS (BUT NOT DECLARED AS CONSTANT SINCE NO INITIAL VALUE EXPRESSION)
- TASK TYPES ARE LIMITED PRIVATE (NO ASSIGNMENT, NO EQUALITY COMPARE)

SIMPLE TASK DECLARATIONS

```
task SIMPLE is          task type ANON is
   . . .                    . . .
end SIMPLE;             end ANON;
                        SIMPLE:ANON;
```

USING TASK OBJECTS IN STRUCTURES

```
AT : array(1..10) of T;--AT(1).E(...);
type REC is             --R.CT.E(...);
  record
    CT : T;
    . . .
  end record;
R : REC;
```

TASKS VIA ACCESS TYPES

```
type REF_T is access T;
RX:REF_T := new T;
```

o REF_T IS NORMAL ACCESS TYPE,
  SO CAN DO ASSIGNMENT AND COMPARE
  TWO OBJECTS FOR EQUALITY

o TO CALL ENTRY E

```
RX.E (...);
```

BINDING TIME OF

NUMBER OF TASKS

1) SIMPLE TASK DECLARATIONS
   NO RECURSION
   => COMPILE / LINK TIME
2) NO ACCESS OBJECTS TO
   OBJECTS OF TASK TYPE
   =>ELABORATION TIME
3) ACCESS OBJECTS USED
   =>DYNAMIC DURING
   EXECUTION

TASK TERMINATION

1) EXECUTE FINAL END =>
   TERMINATE WHEN ALL DEPENDENT
   TASKS ARE TERMINATED

2) TERMINATE ALTERNATIVE IN A
   SELECT STATEMENT

TERMINATE ALTERNATIVE

```
select
  when GUARD_FALSE => ...
  when GUARD_TRUE =>
   terminate;
--ACCEPT PREFERRED
--NO else OR delay CHOICE
```

TERMINATE ALTERNATIVE (CON'T)

1. ALL ENTRY QUEUES EMPTY
2. MASTER IS COMPLETED
3. DEPENDENT TASKS AND SIBLING TASKS ARE:
   A) COMPLETE OR
   B) AT TERMINATE ALTERNATIVE

EXCEPTIONS AND ABORT

<u>abort</u> TASK_NAME_LIST;

- EACH NAMED TASK AND EACH TASK THAT DEPENDS ON THEM "BECOMES ABNORMAL"
- ABNORMAL TASKS MUST TERMINATE PRIOR TO NEXT SYNCHRONIZATION POINT

ABNORMAL TASKS

CALLER RECEIVES "TASKING_ERROR" IF:

1. CALLED TASK IS ABNORMAL
2. BECOMES ABNORMAL DURING RENDEZVOUS
3. BECOMES ABNORMAL WHILE ENQUEUED

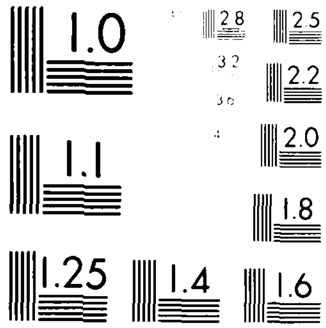
- SERVER TASKS ARE PROTECTED FROM CALLER BECOMING ABNORMAL

SYNCHRONIZATION POINTS

- o END OF ACTIVATION
- o ACTIVATE ANOTHER TASK
- o ENTRY CALL
- o START & END OF ACCEPT
- o SELECT STATEMENT
- o DELAY STATEMENT
- o EXCEPTION HANDLER
- o ABORT STATEMENT

AD-A155 779 AN INTRODUCTION TO ADA (TRADEMARK) FOR SCIENTISTS AND ENGINEERS(U) ARMY MATERIEL SYSTEMS ANALYSIS ACTIVITY ABERDEEN PROVING GROUND MD H E COHEN OCT 83 5/5

UNCLASSIFIED F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART

SHARED VARIABLES

<u>RULES TO FOLLOW</u>

- o IF TWO (OR MORE) TASKS READ OR UPDATE A SHARED VARIABLE --(VARIABLE ACCESSIBLE BY BOTH)
- o BETWEEN SYNCHRONIZATION POINTS
  1) MULTIPLE READERS

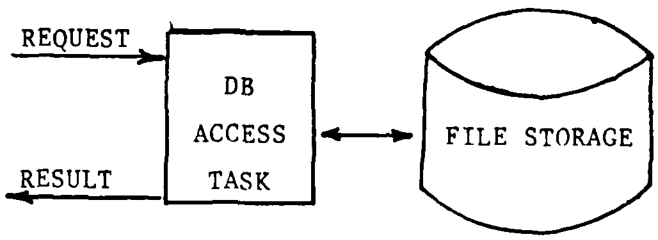
     OR

  2) ONE WRITER

SYNCHRONIZATION POINTS

1) START OF RENDEZVOUS } WITH PARTNER
2) END OF RENDEZVOUS } WITH PARTNER
3) AT ACTIVATION, WITH PARENT
4) AFTER COMPLETION, WITH ALL
5) pragma SHARED (VARIABLE)
   --ONLY SCALERS AND ACCESS
   --ONLY IF R/W INDIVISIBLE
   --EVERY R/W IS SYNCHRONIZED

WHY PRAGMA SHARED ?



READ A STATUS VARIABLE

--EG DATABASE SIZE

--WITHOUT RENDEZVOUS
OVERHEAD

SUMMARY OF TASK TYPES

- o TASKS ARE ONE OF THE FOUR FORMS OF PROGRAM UNIT
- o TASKS ARE A TYPE; (SUBPROGRAMS, PACKAGES, GENERICS ARE NOT)
- o TASKS SUGGEST PARALLEL EXECUTION BY LOGICAL PROCESSORS

LABORATORY #5

LABORATORY #5

OBJECTIVES:

1. WRITE, COMPILE, AND EXECUTE
   A MULTITASK PROGRAM
2. SHOW THAT Ada PROGRAMS
   ARE EASY TO MODIFY

ASSIGNMENT

TRANSFORM EACH PROCEDURE
OR FUNCTION IN YOUR HAND
CALCULATOR PROGRAM INTO A
SEPARATE TASK ACCESSED VIA
AN ENTRY CALL

NEXT PAGE INTENTIONALLY
LEFT BLANK.

TAPE #15

MACHINE DEPENDENT PROGRAMS/SUMMARY

IMPLEMENTATION DEPENDENT FEATURES

- REPRESENTATION CLAUSES
- PRAGMAS
- LENGTH CLAUSES
- ADDRESS CLAUSES
- INTERRUPTS
- INTERFACE TO OTHER LANGUAGES
- UNCHECKED STORAGE DEALLOCATION
- UNCHECKED TYPE CONVERSIONS

PRAGMAS

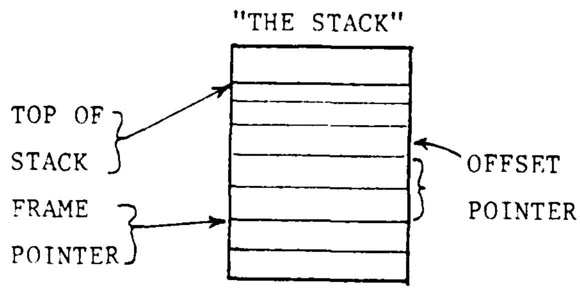
A pragma CONVEYS ADVICE
TO THE COMPILER

```
pragma PACK (TEXT);
pragma INLINE (PROCEDURE_NAME);
```

MEMORY MANAGEMENT

STATIC	-	COMPILE/LOAD TIME
BLOCK ENTRY	-	STACK
DYNAMIC ALLOCATOR (new)	-	COLLECTION IN A HEAP

BLOCK ALLOCATION




```
ADD_A_FRAME :
 FRAME_POINTER := TOP_OF_STACK;
 TOP_OF_STACK := TOP_OF_STACK +
         LENGTH (NEW_FRAME);
```

MACHINE CODE INSERTS

EACH MACHINE INSTRUCTION IS
A RECORD AGGREGATE OF A TYPE
THAT DEFINES THE INSTRUCTION

```
M : MASK;
procedure SET-MASK;
pragma INLINE (SET_MASK);
. . .
SI_FORMAT'(CODE => SSM,
           B => M'BASE REG,
           D => M'DISP);
--IMPLEMENTATION SPECIFIC ATTRIBUTES
```

FORTRAN INTERFACE

```
package FORT_LIB is
  function SQRT (X:FLOAT)
    return FLOAT;
  function EXP (X:FLOAT)
    return FLOAT;
private
    pragma INTERFACE
      (FORTRAN, SQRT);
    pragma INTERFACE
      (FORTRAN, EXP);
end FORT_LIB;
```

```
            package SYSTEM
type ADDRESS is . . . --MEMORY
type NAME is . . . --ENUMERATION
SYSTEM_NAME : constant := . . .
STORAGE_UNIT : constant := . . .
MEMORY_SIZE : constant := . . .
--REPRESENTATION ATTRIBUTES
```

INTERRUPTS (CONT)

- o ABOVE SEMANTICS CAN BE IMPLEMENTED BY HAVING HARDWARE EXECUTE THE APPROPRIATE ACCEPT STATEMENT
- o QUEUED INTERRUPTS MAP TO ORDINARY ENTRY CALLS
- o INTERRUPTS THAT ARE LOST IF NOT PROCESSED CORRESPOND TO CONDITIONAL ENTRY CALLS
- o INTERRUPTS HAVE HIGHER PRIORITY THAN MAIN PROGRAM OR ANY USER TASK

INTERRUPTS

o INTERRUPTS ACT AS ENTRY CALLS
  ISSUED BY HARDWARE TASKS

```
task INTERRUPT_HANDLER is
  entry DONE;
  for DONE use at 16#40#;
end INTERRUPT_HANDLER;
```

RECORD REPRESENTATION (CONT)

```
for PROGRAM_STATUS_WORD use
  record at mod 8;
   SYSTEM MASK        at O*WORD range 0..7;
   PROTECTION KEY     at O*WORD range 10..11;
    -- BITS 8,9 UNUSED
   MACHINE STATE      at O*WORD range 12..15;
   INTERRUPT_CLAUSE   at O*WORD range 16..31;
  end record;
for PROGRAM_STATUS_WORD'SIZE
  use 4*SYSTEM.STORAGE_UNIT
```

RECORD REPRESENTATION (CONT)

```
type PROGRAM_STATUS_WORD is
  record
     SYSTEM_MASK        : BYTE_MASK;
     PROTECTION_KEY     : INTEGER range 0..3;
     MACHINE_STATE      : STATE_MASK;
     INTERRUPT_CLAUSE   : INTERRUPTION_CODE;
  end record;
```

RECORD REPRESENTATION CLAUSE

```
WORD : constant := 4;
type STATE is (A, M, W, P);
type MODE is
  (FIX, DEC, EXP, SIGNIF);
type BYTE_MASK is
  array (0..7) of BOOLEAN:
type STATE_MASK is
  array (STATE) of BOOLEAN;
type MODE_MASK is
  array (MODE) of BOOLEAN;
```

ENUMERATION REPRESENTATION CLAUSE

```
type MIX_CODE is
  (ADD, SUB, MUL, LOA, STA, STZ);
for MIX_CODE use
  (ADD =>1, SUB =>2, MUL => 3,
  LOA => 8, STA => 24, STZ => 33);
```

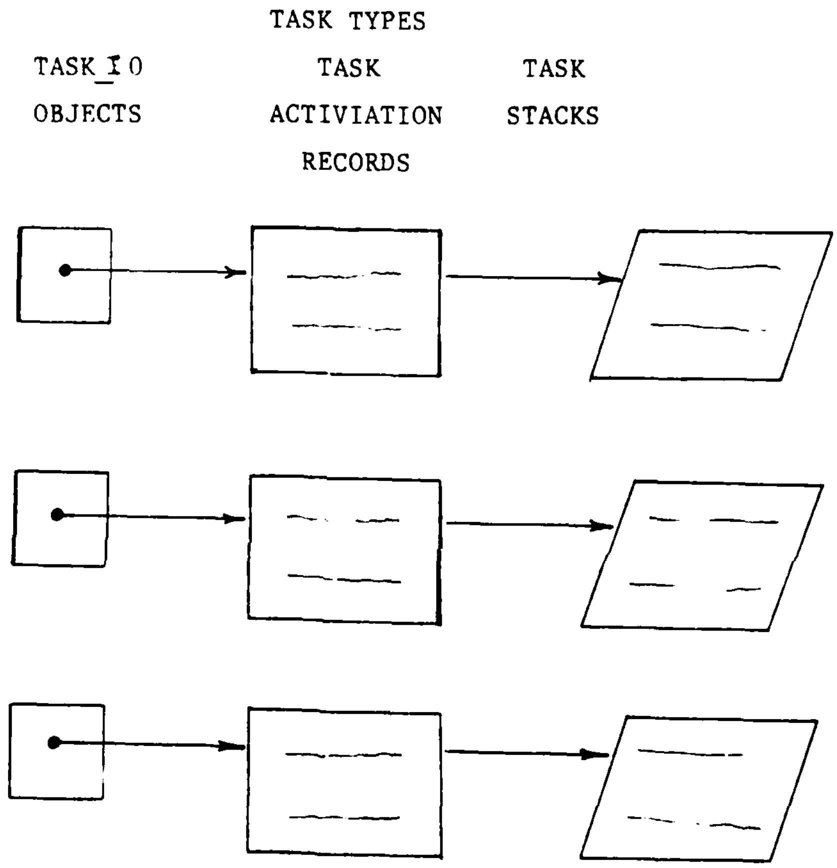
REPRESENTATION SPECS
USED TO CONTROL THE BIT
PATTERNS WHICH REPRESENT
ENUMERATION TYPES
AND
RECORDS
AND THE AMOUNT OF STORAGE
ASSOCIATED WITH A TYPE

SIZE OF OBJECTS
IN A COLLECTION

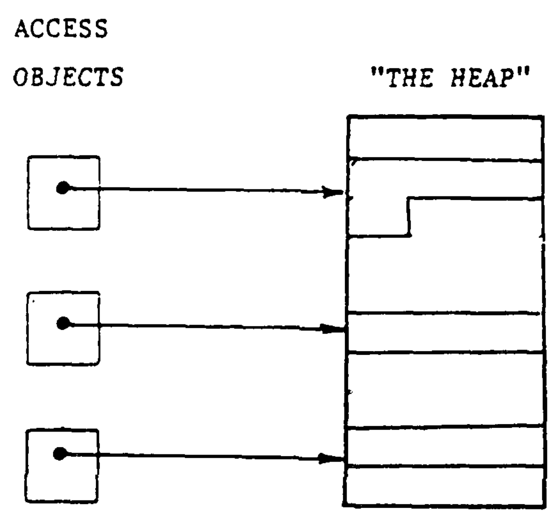
- o ALL OBJECTS MUST BE OF SAME BASE TYPE
- o VARIABLE LENGTH RECORDS CAN OCCUR WITH UNCONSTRAINED DISCRIMINANTS
- o VARIABLE LENGTH ARRAYS OCCUR WHEN BASE TYPE IS UNCONSTRAINED

CONTROLLING MEMORY ALLOCATION

```
for TASK_TYPE_NAME'STORAGE_SIZE
  use INTEGER_EXPRESSION;
 --STORAGE UNITS PER ACTIVATION
for ACCESS_TYPE_NAME'STORAGE_SIZE
  use INTEGER_EXPRESSION;
 --STORAGE UNITS FOR COLLECTION
```


TASK TYPES
TASK_IO
OBJECTS
TASK
ACTIVIATION
RECORDS
TASK
STACKS

HEAP ALLOCATION



UNCHECKED DEALLOCATION

```
generic
   type OBJECT is limited private;
   type NAME is access OBJECT;
procedure UNCHECKED_DEALLOCATION
   (X : in out NAME);
procedure FREE is new
   UNCHECKED_DEALLOCATION
   (OBJ_TYPE,ACCESS_TYPE);
```

UNCHECKED TYPE CONVERSION

```
generic
   type SOURCE is limited private;
   type TARGET is limited private;
function UNCHECKED_CONVERSION
   (S : SOURCE) return TARGET;

function UNSPEC is new
   UNCHECKED_CONVERSION
   (SAC_TYPE, RESULT_TYPE);
```

SUMMARY

SOME Ada IMPLEMENTATIONS WILL ALLOW YOU TO:

- o AFFECT MEMORY MANAGEMENT
- o CONTROL THE REPRESENTATION OF TYPES
- o WRITE INTERRUPT HANDLERS
- o WRITE EMBEDDED MACHINE CODE
- o LINK TO OTHER LANGUAGES

SYSTEM ATTRIBUTES ARE USED FOR IMPLEMENTATION DEPENDENT ALGORITHMS

Ada

REVIEW AND CONCLUSIONS

CONFORMING TO THE Ada STANDARD

(a) TRANSLATE & CORRECTLY EXECUTE LEGAL UNITS, PROVIDED DON'T EXCEED CAPACITY

(b) REJECT UNITS IF EXCEED CAPACITY

(c) REJECT ALL ERRORS AS REQUIRED

(d) SUPPLY ALL REQUIRED PREDEFINED UNITS

(e) NO VARIATIONS, EXCEPT AS PERMITTED

(f) SPECIFY PERMITTED VARIATIONS PROPERLY

CLASSIFICATION OF ERRORS

(a) ERRORS THAT MUST BE DETECTED AT COMPILATION BY ALL COMPILERS

(b) ERRORS THAT MUST BE DETECTED AT RUN-TIME BY ALL IMPLEMENTATIONS. THESE ARE THE PREDEFINED EXCEPTIONS.

(c) ERRONEOUS EXECUTION - RULES THAT MUST BE OBEYED BUT THAT IMPLEMENTATIONS NEED NOT CHECK

(d) INCORRECT ORDER DEPENDENCIES

RUN-TIME PROCESSES

o EXECUTION	:	PROCESS A SEQUENCE OF STATEMENTS
o EVALUATION	:	COMPUTE VALUE OF EXPRESSION
o ELABORATION	:	ACHIEVE EFFECT OF DECLARATIONS, SUCH AS CREATING OBJECTS

ORDER OF EXECUTION

DEFINED

- o IN ABSENCE OF FLOW OF CONTROL COMMAND, STATEMENTS IN TEXTUAL ORDER
- o DECLARATIONS ELABORATED IN TEXTUAL ORDER
- o EVALUATION OF SHORT-CIRCUIT CONTROL

ORDER OF EXECUTION (CONT)

UNDEFINED

- OPERANDS OF AN EXPRESSION EVALUATED IN SOME (UNDEFINED) ORDER
- ORDER OF PARAMETER ASSOCIATIONS
- ORDER IN WHICH in out AND out PARAMETERS COPIED BACK
- ORDER OF TASKS OF EQUAL PRIORITY

:IAL BUSINESS
LTY FOR PRIVATE USE $300

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 12062 WASHINGTON, D.C.

POSTAGE WILL BE PAID BY DEPARTMENT OF THE ARMY

Ada Joint Program Office
3D139 (400 A/N)
The Pentagon
Washington, DC 20301

QUESTIONNAIRE

1. Briefly state your current computer background.

2. Brief comment on course format.

3. How many tapes did you watch at a time?

4. What tapes needed to be replayed for content. Please amplify reasons to replay.

5. Overall evaluation of the course.

6. Do you recommend future courses of this format?

7. What topics do you recommend?

8. Should any tapes be redone - which ones, and why. (Please provide constructive suggestions.)

9. How effective were the lecture notes?

10. What improvement can you recommend for lecture notes.

(If you desire) ______________________________
Name

____

END

FILMED

8-85

DTIC